

12

EUROPEAN PATENT APPLICATION

21 Application number: 88308112.7

51 Int. Cl.4: G06F 15/40

22 Date of filing: 01.09.88

30 Priority: 30.10.87 US 115457

43 Date of publication of application:
03.05.89 Bulletin 89/18

64 Designated Contracting States:
DE FR GB IT

71 Applicant: International Business Machines Corporation
Old Orchard Road
Armonk, N.Y. 10504(US)

72 Inventor: Hernandez, Irene Hernandez
13304 Council Bluff Drive
Austin Texas 78727(US)
Inventor: Ogden, William Craig
Clayallee 242
D-1000 Berlin 37(DE)
Inventor: Sordi, Joseph John
212 Marchmont Drive
Los Gatos California 95032(US)

74 Representative: Hawkins, Anthony George Frederick
IBM United Kingdom Limited Intellectual Property Department Hursley Park
Winchester Hampshire SO21 2JN(GB)

64 System and method for database operations.

57 A data base manager system operates a relational data are stored in a memory. The system sequentially generates a plurality of prompt screens on a display. Operator responses thereto, Input at keyboard, define a query statement which, when executed by a processor retrieves a correlative data subset from the memory. The query is comprised of clauses. A plurality of sequential prompt screen-responses specify at least one of the clauses. Available valid response choices and/or response instructions displayed in a screen automatically appear and are varied as a function of response to the prior screen. In one embodiment, the left side, comparison operator, and right side of a row condition clause are each specified by at least one prompt screen-response. When the left side or right side is complex including a summary function or expression, a sequential plurality of prompt screen-responses specifies the respective left or right side. The system enables a database user to specify complex query statements without knowledge of the syntax and semantics of the query language.

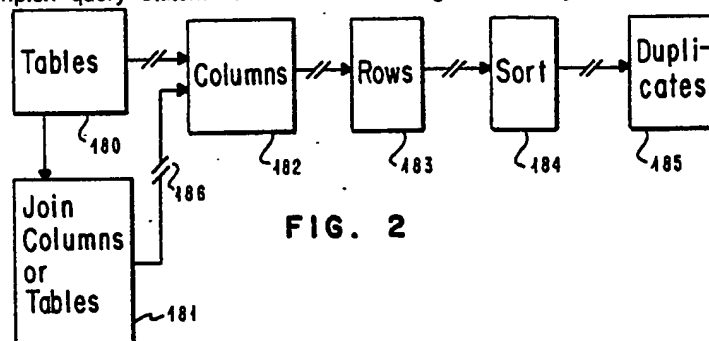


FIG. 2

SYSTEM AND METHOD FOR DATABASE OPERATIONS

This invention relates to database operations and, more particularly, to the formulation of search commands by an operator for retrieving selected data therefrom.

In the computerised database field, vast quantities of data are stored in computer systems, a greatly
 5 simplified but nevertheless illustrative schematic example of which might be information relating to a manager's staff as depicted in Fig. 1.

For a database to be useful, the ability to retrieve only a selected subset of the data satisfying a correspondingly unique set of conditions is highly desirable. As but a simple example, the manager may wish to obtain a list from the aforementioned database of only those employees' names starting with "W"
 10 and their i.d. numbers.

In order to meet this need, various computerised systems have been developed for retrieving such information. These systems typically include a computer program, commonly referred to as a database manager, for storing, editing, updating and retrieving data in response to various commands entered through a user interface. Specifically, with respect to retrieval of data, numerous computer languages were
 15 devised for formulating search commands or "queries" to which the database manager was responsive in providing the requested data. These queries were basically search instructions encoded so as to cause a computer and associated database manager to carry out the desired search.

Several problems have been associated with these languages for formulating database queries. First, many of the query languages differed from conventional programming languages. The database user with
 20 programming experience was thus required to learn an entirely new set of commands just to get meaningful data out of the database. Users without such experience, such as many terminal operators who are also without computer experience of any kind, were thus forced to learn a form of computer programming just to interact with the database. Moreover, such query languages required knowledge of highly complex syntax and semantics rules, thereby further limiting the numbers who could successfully retrieve data to only a
 25 highly and expensively trained few. This, in turn, adversely affected the utility of such computer systems and seriously inhibited their use by a widespread number of individuals.

A representative such query language is the Standard Query Language or "SQL" detailed in the Draft Proposal, ANA Database Language SQL, Standard X3.135-1986, American National Standard Institute, Inc., 1430 Broadway, New York, New York 10018. Detailed discussion of SQL is also set forth in "IBM Database
 30 2 SQL Reference" Document Number SC26-4346-3, IBM Corporation.

Efforts were made to simplify the knowledge required to formulate a database retrieval command, thereby rendering access to the database more "user friendly". This gave rise to several techniques such as less formal fourth level, non-procedural, or "natural" query languages. Search commands derived therefrom had many attributes of natural speech patterns and thus were easier to formulate. Other
 35 approaches to assist less technically sophisticated users in query formulation included query-by-form, query-by-example, various user aid help menus, and the like, wherein operator responses were sought to more-or-less user-friendly graphics interfaces.

Whereas such techniques were helpful to some degree in assisting users in formulating search statements, they were nevertheless limited to relatively straightforward and simple search criteria as the
 40 hereinbefore noted example. When more sophisticated queries were desired, such as a listing from Fig. 1 of all employees having a salary plus commission between \$20,000-\$40,000, they could not be readily constructed without a more detailed knowledge of the syntax and semantics of the query language, thereby once again seriously restricting utility of the database.

Thus, in summary, less-trained operators were unable to use complex query languages. On the other
 45 hand, techniques for reducing the required syntactic and semantic knowledge thereby limited the complexity of the queries which could be constructed and executed.

Techniques have long existed for assisting operators in providing proper commands to computers, often taking the form of help screens or menus which provide command information and choices to the user during program execution. However, typically some knowledge of the command languages was required of
 50 the user.

U.S. Patent 4,506,326, entitled "Apparatus and Method for Synthesizing a Query for Accessing a Relational Database" discloses a more user friendly graphic query in query by example (QBE) syntax wherein a linear query is synthesized from the graphic query input at a user terminal.

IBM Technical Disclosure Bulletin, Vol. 27, No. 9, February 1985, page 5140, entitled "Relational Database Features for Natural-Language Interface Grammars" details mapping general natural-language

terms into corresponding formal database query languages.

IBM Technical Disclosure Bulletin, Vol. 25, No. 11A, April 1983, page 5499, entitled "Model Query Generation" discloses a mechanism giving the user capability to specify certain query operations without knowledge of a query language.

6 Several commercially available software database products embodied several techniques for enabling the user to query the database with little knowledge of the query language. A product sold under the product name "PC Focus" is available from Information Builders Incorporated, 1250 Broadway, New York, New York 10001. Other such products include those sold under the products name "Easy SQL" (IBM Corporation, Old Orchard Road, Armonk, New York 10504); "Paradox" (Ansa Software, 1301 Shoreway
10 Road, Belmont, California 94001); and "dBaseIII+" (Ashton-Tate, 20101 Hamilton Avenue, Terrence, California 90502).

It is an object of the present invention to provide a simplified system and method for operator formulation of complex query statements executable by computerised databases.

According to the invention, there is provided a method of operating a computer system for generating a
15 search command clause for accessing a database comprising the steps of: generating an displaying a first prompt relating to a first component of the clause; registering a response to said first prompt; generating and displaying a second prompt relating to a second component of the clause in response to the registered response to said first prompt; and registering a response to said second prompt, wherein said registered responses comprise required components of the search command clause.

20 There is also provided a data processing system for database operation comprising means for generating and displaying a first prompt relating to a first component of a search command clause for the database, means for registering a response to said first prompt, means for generating and displaying a second prompt relating to a second component of the clause and means for registering a response to said second prompt, wherein said registered responses comprise required components of the search command
25 clause.

In an embodiment of the invention to be described in detail later, an arrangement for generating operator-specified complex query statements is shown. A query is comprised of a plurality of clauses. For each clause, at least one prompt screen is displayable requesting operator response(s) in order to specify the clause. Each screen may include a plurality of possible valid response choices satisfying the requested
30 response, whereby the operator selects the appropriate prompted choice. Alternatively, the operator may key in a proper non-listed response based upon instruction provided in the screen(s).

For some clauses, a sequence of prompt screen-operator responses result from sequential display of a plurality of prompt screens and recording of responses, thereby defining the respective clause. Available valid response choices and/or response instructions displayed in a screen automatically appear and vary as
35 a function of response to a prior window. Upon operator completion of requested responses for all prompt screens associated with a clause, one or more prompt screens requesting information for defining the next clause are automatically displayed.

The left side, comparison operator, and right side of a row condition clause are each specified by at least one prompt screen-response. When the left side specification is completed, a different prompt screen
40 for the conditional operator automatically appears displaying a different set of valid response choices which vary as a function of response(s) to the left side prompt screen(s).

Prompt screen-operator responses are provided for specifying a row clause of a query wherein the left side or first component of the clause may be complex. A first prompt screen is displayed for the first component and operator response thereto is recorded. A second prompt screen for the component is
45 thence automatically displayed which varies as a function of the particular response to the first prompt screen. Valid response choices to the first screen include a prompt for a summary function or an expression. If the summary function or alternatively the expression choice is selected by the operator, the second prompt screen automatically displayed requests information necessary to specify the summary function or the expression, respectively. Thus, a sequential plurality of prompt screen display-responses is
50 automatically provided for defining a complex first component of a row clause.

The embodiment of the invention will now be described, by way of example, with reference to the accompanying drawings, in which:

Fig. 1 is an illustration of a representative database table;

Fig. 2 is a simplified flow diagram depicting steps in formulating a query statement and the clauses
55 thereof;

Figs. 3A, 3B, 3C, 3D, and 3E are a representative sequence of prompt screens including operator responses for formulating a query;

Fig. 3F is a data output table from the table of Fig. 1 after execution of the query formulated in Figs. 3A-3E;

Figs. 4A, 4B, 4C, and 4D are a representative sequence of prompt screens including operator responses for formulating another type of query;

Fig. 4E is a data output table from the table of Fig. 1 after execution of the query formulated in Figs. 4A-4D;

Figs. 5A, 5B, 5C, 5D, 5E, 5F, 5G, and 5H are a representative sequence of prompt screens including operator responses for formulating yet another type of query;

Fig. 5I is a data output table from the table of Fig. 1 after execution of the query formulated in Figs. 5A-5H;

Figs. 6A, 6B, 6C, 6D, 6E, and 6F are a representative sequence of prompt screens including operator responses for formulating yet another type of query;

Fig. 6G is a data output table from the table of Fig. 1 after execution of the query formulated in Figs. 6A-6F;

Fig. 7 is a simplified flow diagram depicting steps in formulating a row clause of a query statement;

Figs. 8A, 8B, 8C, and 8D are more detailed flow diagrams of the query statement formulating steps shown in the flow diagrams of Fig. 2 and Fig. 7;

Fig. 9 shows the mapping of clause nomenclature of a representative complex query language (SQL);

Fig. 10 shows the detailed syntax and semantics of a representative complex query language SQL SELECT statement; and

Fig. 11 shows a schematic representation of the computerised relational database system of the present invention.

The present invention provides a user interface for formulating complex search instructions or "query" statements which are useful in retrieving selected information from computer databases. The purpose of this system is to provide a user-friendly interface whereby minimal knowledge is required from the end user with respect to syntax and semantic constraints of the associated query language in order to properly formulate a query. The embodiment disclosed is related to a relational database product.

In operation, an operator provides a query to the relational database program in order to obtain a subset of the information stored in the database. For example, the operator needs a report listing all employees' names starting with "W" and their i.d. numbers from a database represented by Fig. 1. Those knowledgeable in the use of the high level query languages such as the hereinbefore noted Structured Query Language (SQL) are able to provide a single instruction to perform this function. However, those not familiar with SQL would not easily be able to provide the instruction in accordance with the proper syntax and semantics to accomplish this task.

The embodiment of this invention provides a system for sequentially prompting the user through the various elements of the syntax and semantics of a complex query statement without requiring prior knowledge thereof. More particularly, a system is disclosed wherein a query is broken into individual clauses each of which is then defined by user response to sequential prompts. Complex clauses of the query are defined by user response to a plurality of sequential prompts for the particular clause, thereby obviating the user's need for syntax and semantics knowledge of the particular query language.

First, with reference to Figs. 1 and 2, an overall view will be given of the elements or clauses comprising a query statement. Next, several examples in Figs. 3-6 will be discussed which illustrate formulation of actual query statements as well as the resultant output data sets. This will then be followed by a discussion, with reference to Figs. 7-8, of a software procedure for prompting formulation of a database query according to the present invention. The procedure may be implemented in one embodiment with reference to the representative flow chart disclosed in Fig. 8. With references to Figs. 9-10, in order to more clearly appreciate the significant benefits of the query prompting system of the present invention over other conventional methods, a discussion will be provided in greater detail regarding alternative construction of a query statement by means of the SQL query language. Finally, apparatus according to the invention for generating queries will be discussed with reference to Fig. 11.

Fig. 2 is a pictorial view of a sequence of clauses 180-185 comprising a query, each of which may be sequentially defined by means of operator responses to user interface screen prompts in order to formulate a complete query statement. It will be recalled that the completed statement will be uniquely functionally related to the particular subset of data to be retrieved from the database and will operate as a command statement to which the database manager software program is responsive in retrieving the desired data.

As an example of performance of the steps schematically indicated in Fig. 2 to formulate a representative query, let it be assumed with reference to Fig. 1 that it is desired to retrieve from the Staff table

depicted therein a list of all individuals whose name commences with "W" and their corresponding employee i.d. number. Let it further be assumed that more than one table may exist in the database as, for example, a second table listing department numbers and locations. Inasmuch as the department number is common to both this latter table and the Staff table, data between the two tables may be interrelated as
 5 desired giving rise to the previously noted term "relational databases". Thus, for example, the location of an employee may be obtained by interrelating the two tables wherein the particular employee's department number may be determined from the Staff table of Fig. 1, and then by querying the second table for location of the department having this department number, the employee's location may be determined.

With continuing reference to Fig. 2, clause 180 is intended to indicate that, as a first step in structuring
 10 or formulating the desired query, it may be necessary to select more than one table from the database (as just described) and thence to join selected columns within the tables or the entirety of selected tables in a cartesian product as shown by clause 181. For simplicity in the example presently under discussion, however, such a joiner is not required in that all the necessary information to respond to the query appears in the Staff table of Fig. 1.

Proceeding to the "columns" clause definition, 182, it will be necessary to formulate an appropriate portion of a computer instruction comprising a query which will retrieve only the desired employee name and i.d. number columns from the Staff table of Fig. 1. However, if the query statement was limited only to such information, the entirety of the i.d. and name columns of Fig. 1 would be retrieved rather than limiting retrieval as desired to only those names and corresponding i.d.'s wherein the names fit the aforementioned
 20 criterion of beginning with a "W". Accordingly, the rows clause, 183, represents that portion of the completed query which, when executed, will delimit the number of names retrieved from the name column to only those fitting the prescribed condition, namely only those names commencing with the "W". Once the names retrieved from the name column have thus been parsed by the row condition specified by clause 183, it may be desired to have the remaining rows which met the row condition sorted in some fashion such
 25 as alphabetically or in ascending or descending numerical i.d. order. Thus, a sort clause, 184, portion of a complete query may be formulated.

Finally, in some instances, notwithstanding selected retrieval of only a subset of data comprising the database by reason of the operation of the previously described portions of the query statement, duplicate data fields may nevertheless be retrieved. If it is desired to eliminate such duplicate data outputs after the
 30 database is queried, a duplicates clause 95 of the completed query statement may be included to suppress retrieval of such duplicates. The breaks (//) 186 in Fig. 2 denote points on the prompting sequence wherein the user can either continue through or bypass the next step in a sequence. An example of this might be a query for all columns within a given table wherein limiting the rows, sorting, and eliminating duplicates may not be desired, in which case formulating clauses, 183-185, are unnecessary.

Figs. 3A-3E depict a series of actual computer terminal screen prompts to an operator which were generated by implementation of a software program of the present invention hereinafter discussed with reference to the flow diagrams of Figs. 7-8. Also depicted are proper operator responses thereto, resulting in formulation of a completed query statement to a database, the desired data output table of which is shown in Fig. 3F (i.e., the listing of employee names commencing with "W" and correlative i.d. numbers).

First, in Fig. 3A, the operator is prompted by means of a tables window, 187, for the name of tables in the database to be queried, whereupon the name of the Staff table is properly entered, reference numeral 188. In Fig. 3B, it will be noted that in accordance with the flow diagram of Fig. 2, the operator is then automatically prompted by means of columns window, 190, for information comprising the columns clause portion of a complete query statement whereby the number of columns and their functional makeup may be specified. It will be noted that in the columns window, 190, a scrollable list of available columns names 191 is provided for operator selection and that the operator has accordingly properly selected the column names I.D., 192, and NAME, 193. Operator response or selections will be hereinafter designated in the accompanying figures with a caret so as to not require further comment except as indicated. These column names 191 will be those of the table selected by clause 180 or a combination of columns of a plurality of
 50 tables joined by means of clause 181 of Fig. 2.

In addition to mere selection from tables of existing columns, provision is made for specifying new columns which may be mathematical functions of one or more existing columns as in the case of an expression 194 (such as salary + commission) or a summary function 195 (such as the average or minimum of a specified column). If the operator specifies that an expression 194 or summary function 195
 55 type of column clause is to be defined, one or more separate prompt screens requiring operator response (to be discussed later but similar in function to those of Figs. 3A-3E) will be displayed.

Next, with reference to Fig. 3C, a rows prompting window 196 will automatically be displayed upon completion of the columns clause 182 of Fig. 2. This prompts operator response so as to define the desired

row condition or clause 183 of Fig. 2 which restricts retrieval of rows in the NAME column only to those commencing with "W". Without the rows condition 183 portion of the query, all names and corresponding employee i.d.'s would accordingly be retrieved as the selected columns from the staff table. Out of all such possible rows, it is of course desired to limit them whereby in order for names rows to remain in the parsed
 5 retrieved list (and their corresponding i.d. numbers) they must satisfy the following row condition:

**"If the name is starting with W" (then
 retrieve it and list it as data output).**

From the foregoing it will be noted that a row condition or clause is comprised of three parts each separately underscored which may be clearly seen in Fig. 7. The first underscored part may be referred as to the first component or left side 280 of the row condition. The second singly underscored portion may be referred to as the second component or comparison operator 281, and the third underscored portion may
 15 be referred to as the third component or right side 282 of the row condition. It will further be noted that the second component is comprised of two subcomponents, namely the doubly underscored portion referred to as the comparison and the remainder or verb. Both may be collectively referred to as the comparison operator as noted.

Referring again to Figs. 3C-3E, in Fig. 3C the operator is being prompted by means of the column names 197 portion of the row condition window 196 to specify the first component or left side of the row condition (i.e., "If the name..."). As shown in Fig. 3C, the operator has correctly selected the "NAME" column name as designated by the caret, whereupon the second component of the row condition is automatically prompted for as shown by appearance of the comparison operator window 200 of Fig. 3D. In
 20 response to this prompt the operator has selected the appropriate verb 201 ("is") as well as the proper comparison 202 ("starting with...") so as to complete the second component of the row clause 183 of Fig. 2 for the desired query.

Finally, with respect to the Fig. 3E, upon completion of entry of the second component of the row condition, the operator is automatically prompted for completion of the row condition by providing an appropriate response to a request for information to complete the right side or third component of the row condition. Such prompting may be seen by the automatic appearance of the "starting with..." prompting
 30 window 203 upon the operator's completion of response to the comparison operator's window 200 of Fig. 3D. In Fig. 3E it will be noted that the operator has correctly input as a response to "starting with..." the correct right side condition "W", thereby completing the entire query statement (inasmuch as sorting and duplicates clauses 184 and 185 are not desired in the present example).

A complete query statement has thereby been formulated by an operator which, when executed by the computerised database manager, will result in the desired data output table shown in Fig. 3F. To put in perspective the ease with which query statements may be formulated by the system of the present invention in comparison to other methods, an equivalent query statement for this example in the SQL
 40 language is of the form

"SELECT ID, NAME FROM Q.STAFF WHERE NAME LIKE 'W%'".

This statement may be input directly by means of keyboard entry. However, it is readily apparent that in order to do so the operator must have knowledge of the required format of the statement particularly in
 45 terms of syntax and semantics.

A convenient feature of the system being described is that not only are appropriate subsequent prompting screens automatically presented for corresponding operator response to define each subsequent query clause, but the operator's responses are cumulatively echoed back on the left-hand side of each screen Fig. 3A-3E so as to provide a continuous record of where the operator is in construction of the
 50 query. Accordingly, once the correct Staff 188 response has been made to the tables window 187 (Fig. 3A), this response is echoed at 189 in the screen of Fig. 3B so as to show that the Staff table has been selected. (The echoed "Q" is merely a user i.d.) Similarly, upon correct selection of I.D. and NAME columns 192 and 193, respectively, in the columns window 190 of Fig. 3B, these column selections are automatically echoed on the left-hand side of the screen of Fig. 3C as indicated by reference numeral 204. In like manner, the
 55 proper first component and comparison operator or second component of the row condition clause may be seen echoed at reference numeral 205 in the screen of Fig. 3E after proper selection from the column names window 197 (Fig. 3C) and comparison operator window 200 (Fig. 3D).

Yet an additional aspect of the example of Fig. 3A-3E must be noted, namely that as with the specification of column names (Fig. 3B) from the tables, an expression 198 or summary function 199 (Fig. 3C) may be specified as a first component or left side condition of a row clause, although in the present example only the column name "NAME" was selected. Moreover, the third component or right side of a row clause condition need not be a single value such as the alpha character selected in Fig. 3E or a constant, but rather may be a complex expression as desired.

An example wherein an expression 198 may be desired as the first component of a row condition might be, in the present example, "salary + commission". In such a case it might be desired to retrieve from the Staff table of Fig. 1 a list of all names wherein this sum of two columns meets a specified condition (such as where the salary + commission might be greater than a constant, as will be hereinafter discussed in greater detail with reference to the example of Figs. 4A-E).

In like manner, an example wherein a summary function 199 (Fig. 3C) might be desired as a first component of a row condition or clause might be one in which it is desired to list minimum and maximum salaries in each department wherein the particular department's average salary must first meet some condition such as being less than a specified value. This latter situation will be described in greater detail with reference to Fig. 5A-H wherein the row condition to retrieve such data is "avg(salary) less than 18,000" and wherein it will be noted that the first component condition is complex and not merely a column name but rather a function of a column name.

Although in the example of Fig. 3A-3F a simple column name and value are selected to specify the first and third components of the row condition, it is a feature of and contemplated by the invention that in a general case the first and/or third components of the row clause may be complex and thus may combine two or more columns or be functions of one or more columns wherein complex first and third components result. It is a feature of the invention to automatically make the operator aware of these possibilities by means of the prompt such as those at 198, 199 (Fig. 3C) or 232 (Fig. 4D).

Moreover, it is a further feature of the invention to automatically provide for one or more specific sequential prompt screens after the operator elects to define a complex function for the first or third components of a row condition as, for example, in the selection of the expression 198 or summary function 199 choices in the column names window 197 of Fig. 3C. These prompt screens will seek appropriate responses from the operator so as to assist in the definition of these complex row condition components without requiring of the operator syntax and semantic knowledge to construct these components correctly.

Fig. 7 provides an overall illustration of the sequence of steps in prompting for a row condition in accordance with the present invention, the details of which will be discussed with respect to the flow diagram of Figs. 8A-C. A series of steps and associated prompt screens and operator responses occur between location 115 (Fig. 8B) and location 170 (Fig. 8C) in the flow diagrams. The series of steps including prompt screens and responses for specifying a complex first component or left side 280 of the row clause (at 122, Fig. 8B) may be seen as the steps of selecting a column 206 (steps 123-141, Fig. 8B), defining an expression 207 (steps 125-141, Fig. 8B) which includes a screen prompt 209 for the expression (step 126, Fig. 8B), or selecting a summary function 208 (steps 132-141, Fig. 8B). The latter, as in the case of defining an expression, further includes prompting 210 (step 133, Fig. 8B) for the summary function and then defining the argument therefor (step 135, Fig. 8B) by selecting a column 283 (step 136, Fig. 8B) or defining an expression 284 in response to a prompt 285 for the expression (step 139, Fig. 8B). In Fig. 3C, a representative prompt screen may be seen for the left side row condition prompting the operator to select the column 206, define an expression 209, or select a summary function 208. Operator input to complete specification of the left side may also be seen in Fig. 3C. Representative prompt windows for defining an expression 207 may be seen in Figs. 4A-4B as well as the appropriate operator response to complete defining this left side of the row. In like manner, representative prompt screens may be seen in Figs. 5D-F for prompting an operator to select a summary function 208 as well as a proper operator responses thereto in order to complete defining this type of left side row condition.

Still referring to Fig. 7, steps required to define the second component or comparison operator of a row clause may be seen schematically depicted therein generally at 281 corresponding to the more specific steps 151-151A represented in the software flow diagram of Fig. 8C. Representative prompt windows for an operator responses to conditions thereby specifying this second component of a row condition may be seen with reference to Figs. 3D-E (in the case of a left side column), Figs. 4C-D (for expression 207 specification), and Figs. 5G-H (for left side summary function 208 definition).

Finally, still referring to Fig. 7, the prompting steps and operator responses thereto in order to specify the right side of a row clause, corresponding to steps 151B, 156-158 of the software flow diagram of Fig. 8C) may be seen generally represented as reference numeral 282. Valid responses to prompts for the comparison operator 281 may include "Equal to", "Greater/less than", "Between two values", and

"Starts/ends/contains", as well as a null condition. Dependent upon which comparison operator is thereby specified in the prompt-response step 281 defining the second component of the row, a correlative one of the prompt-response steps 286, 287, 288, and 289 will be completed thereby specifying the right side 282 of the row clause and completing the clause in its entirety. A representative prompt window and response for this right side 282 completion may be seen in Fig. 3E (with respect to a left side column 206), Fig. 4D (for a left side expression 207), and Fig. 5H (for a left side summary function 208). Upon completion of the specification of the right side 282 of the row condition, the computerised prompted query program of the present invention will be at reference numeral 170 of Fig. 8C.

Figs. 4A-E illustrate a feature of the present invention just discussed wherein a complex first component of a row clause is prompted for. In this example, with reference to Fig. 1, it may be assumed that a manager desires to retrieve from the database of Fig. 1 a listing of all employees, employee I.D. numbers, and salaries but to limit this employee list to only those fulfilling the row clause condition that their salary + commission is greater than \$18,000. The equivalent SQL select statement or search command is "SELECT I.D., NAME, SALARY FROM Q.STAFF WHERE SALARY + COMM GREATER THAN 18000".

From the sequence of prompts and corresponding responses shown in Figs. 4A-4E, it will once again be understood how easily such a query statement may be prompted for without requiring the operator to know what the form in which the previously shown SQL statement must appear for proper execution by the database.

In Fig. 4A it will be noted that, similarly to the previously described example of Fig. 3A-E, the operator has already correctly input, in response to prior prompt screens for table and column selection, the appropriate table to query (Q.STAFF) as well as the desired columns (I.D., NAME, and SALARY). Also, as with the prior example, these selections have been echoed to the left side 220 of the prompt screen. However, a significant difference from the prior example may be noted in Fig. 4A, namely that when the row condition prompt window 221 appears, rather than simply selecting a listed column from the column name's window 222 as the left side or first component of the row condition, the operator has indicated at 223 that it is desired to specify a complex first component, namely an expression, (i.e., salary + commission). As has been previously noted, with prior systems for assisting the less-trained user to construct query statements, the user was not even made aware that it was possible to construct such complex first or third components of row conditions during specification of the row clause, but rather was lead to assume that only preselected and predefined items such as column names were available.

Referring to Fig. 4B, once the desire to specify a left side row expression has been entered, an expression window 224 prompt automatically appears prompting the user to input a desired complex expression. At reference numeral 225 the operator has accordingly correctly keyed in the desired first component of the row clause, namely SALARY + COMMISSION. Upon entering this expression, with reference to Fig. 4C the expression is echoed at 226 to the left side of the prompt screen and the selection of comparison operators appears in the comparison operator window 227 automatically. In response to prompting for the necessary syntactically and semantically correct responses to specify the second component or comparison operator for the row condition, it is evident at 228 and 229 that the operator has correctly selected the verb "Is" and comparison "Greater than", respectively.

Throughout this disclosure items appearing in windows which are valid and thus selectable are designated by a bullet or caret whereas those which may be syntactically or semantically invalid and thus not available for selection may nevertheless be displayed in the window. (Absent the bullet or caret of course.) However, if the operator attempts to select an invalid item, an error statement will appear requesting re-entry of another item. It is accordingly a feature of the invention to so limit the choice of items available for entry in an automatically appearing subsequent prompt window to only those which are syntactically and semantically valid as a function of a prior entry.

Thus, with reference to Fig. 4C, it will be noted that because the previously specified first component of the row condition was numeric in nature (i.e., salary and commission), comparison operators associated with alpha characters or strings such as "starting with", "ending with", "containing" at reference 230 are not valid comparison operators which may be entered as part of the query statement at this point (notwithstanding that the operator may nevertheless attempt to select such operators).

Referring to Fig. 4D, it will again be noted that the operator selections resulting from the previous comparison operator prompt window 227 (i.e., "Is greater than") is reflected at the left side of the prompt screen at location 231 and that a "Greater Than" prompt screen 232 thereby automatically appears to prompt the operator for completion of the row clause by specifying the third component or right side thereof. Once again, a feature of the present invention is thereby illustrated wherein in the specifying of a

row clause, a prompt window will appear having contents functionally related to the nature of a prior response to a prior prompt window. In other words, when the operator indicated completion of a specification of the left side of the row condition by hitting an enter key or the like, the prompt window of Fig. 4C automatically appeared prompting the operator to specify the second component of the row clause.

8 Similarly, once the operator designated completion of the task of specifying the second component or comparison operator of the row clause, the "Greater Than" screen prompt 232 of Fig. 4D appeared which was selected for display as a function of the particular comparison operator selected in the comparison operator window 227 of Fig. 4C. By means of the "Greater Than" window 232, the operator is thereby prompted to recognise that the third component or right side of the row condition may be completed by
10 entering a value, column name, or expression.

In the example under discussion, the operator has thus correctly entered the value 18000 at reference numeral 233, thereby completing formulation of the row clause.

If a different such comparison operator was selected from the comparison operator screen 227 (Fig. 4C), a correlative differing prompt window would appear in the screen of Fig. 4D prompting the user for
15 inputs to complete a row condition uniquely associated with that particular comparison operator. An illustration of a different right side condition prompting window appearing in functional response to the particular comparison operator selected may be seen with reference to the previously described example of Fig. 3A-E, wherein in Fig. 3D in response to selection of the "Starting With" comparison operator, the Starting With prompt window 203 of Fig. 3E appears.

20 Once the complete query statement has thereby been defined by the operator by means of responses to the query windows of Figs. 4A-D, this formulated query statement may thence be executed by the database manager, resulting in the data output table shown in Fig. 4E. It should be apparent that the fields in the output table are a subset of the table of Fig. 1 satisfying the desired data retrieval condition associated with the query statement or search command, namely that a list was desired of all individuals,
25 names, their i.d. numbers and salaries from the table wherein their salaries + commissions exceeded \$18,000.

The series of prompt screens and corresponding operator responses shown in Figs. 5A-H illustrate yet another situation in which a complex component of a row clause query may be automatically prompted for. In the preceding example an expression was prompted for as a first component of the row condition,
30 however, in the instant example it will be shown how a summary function may be prompted for as a complex first component.

In this example, it will be assumed that it is desired to retrieve from the Staff table of Fig. 1 a list of departments from the staff with an indication of the minimum and maximum salaries of individuals within each department, and wherein the rows of the output table are further constrained to showing these
35 department numbers, minimum, and maximum salaries only for such departments having an average salary less than \$18,000. A typical complex query language statement such as that of SQL would accordingly take the form

"SELECT DEPT, MIN (SALARY) MAX (SALARY) FROM Q.STAFF GROUPED BY DEPT HAVING AVG (SALARY) LESS THAN 18000".

40

From a review of the prompt screens and responses associated with Figs. 5A-H, it will again be apparent how easy it is for an untrained operator to arrive at such a complex query statement without syntax and semantics knowledge simply by following the instructions of each prompt screen.

45 As with the case of the examples for Figs. 3A-E and 4A-D, a sequence of screen prompts will appear (not shown) for specifying the Staff table and the dept column out of the Staff table. The operator's appropriate responses to these prompts are reflected in the left hand side of the prompt screen shown in Fig. 5A. Inasmuch as additional columns are desired for output which are summary functions of columns, in the columns prompt window 232 the summary function 233 is selected by the operator in order to define
50 summary function columns. Next referring to Fig. 5B, the Summary Functions window prompt 234 will automatically appear due to selection of the Summary Functions feature at 233 in Fig. 5A. In Fig. 5B, the operator has selected as Summary Function items the functions Minimum and Maximum shown at reference numeral 235.

In Fig. 5C it will be noted that in response to operator selection of these Minimum and Maximum
55 functions 235, a Summary Function Items prompt window 236 will appear prompting the operator to select a column or expression as an argument of the selected Min and Max summary functions (which have, consistent with prior echoes, then echoed to the left side of the prompt window at 237). The operator has selected "Salary" at 238 as the argument for the Min and Max functions 237 as shown in Fig. 5C.

Figs. 5D-H depict the sequence of prompt windows and correlative operator responses so as to define the complex first component of the row condition wherein "AVG(salary) is less than 18000". The function of this row clause is to cause the retrieval from the database of department numbers and the minimum and maximum salary within each department, but only for such departments meeting the row condition criteria that the department must have an average salary of employees within the department of less than \$18,000. Again, it will be recalled that without the prompt window sequence for summary functions (in like manner to the prompt sequence for an expression in the example of Fig. 4), the operator frequently would have no way of knowing (without knowledge of a complex query language) that more complex queries having complex row conditions could be made. Rather, the operator would assume that for example the left side conditions of a row condition would be limited only to those columns previously specified during construction of a columns clause of a query statement. Moreover, even if such an operator was aware that complex first and third components of a row clause could be used to obtain more sophisticated database queries and resultant data output tables, without the prompt window sequences of the present invention an operator without knowledge of the particular complex query language would not know how to formulate such components.

Returning to Fig. 5D, when the operator has provided an Indication 239 that a summary function is desired as a left side row condition, the summary functions prompt window 240 automatically appears (Fig. 5E) wherein the operator has selected the AVG function as shown by the caret 241. In Fig. 5F, upon selection of the AVG function (echoed at 242), the Summary Function Items prompt window 243 automatically appears prompting the operator for an argument for the AVG summary function (which may be either a column or an expression as indicated in the window 243). Again, without such prompting, in addition to not knowing that a summary function or expression might be available as a left side row condition, the operator would further not know that the argument of a summary function might, in addition to a column, equally as well be an expression which might be defined during the specification of the complete row clause.

After the operator has selected Salary 244 in a prompt window of Fig. 5F as the argument to the AVG 242 summary function item, this argument is echoed at 245 in the prompt window of Fig. 5G and the Comparison Operator's prompt window 246 automatically appears. As with the prior examples, this Comparison Operator window 246 will prompt the operator for selection of one of a plurality of displayed available operator choices, permitting selection of only syntactically valid ones designated by the bullets. For example, if the operator sought to select the operator "Ending with" 247 an error flag would be displayed requesting selection of another operator. The reason for this is that the "Ending with" operator 247 is syntactically compatible with character data and not numeric data such as the mathematical function AVG.

Still referring to Fig. 5G, when the operator has responded to the Comparison Operator's window 246 by selecting the appropriate verb "is" 248 and comparison "Less Than" 249, the corresponding "Less Than" prompt window 250 automatically appears (Fig. 5H) requesting the operator for a response providing the third component or right hand side of the row clause to thereby complete it. The selection in Fig. 5G of "Less Than" is of course echoed at 251 in the prompt screen of Fig. 5H. It will further be noted in Fig. 5H that by reason of the "Less Than" comparison operator window 250, the operator is advised that the row condition right side may be either a column name or an expression and thus is not limited only to predefined columns appearing in a list. In the window 250 it is clear that in response to the prompt information contained in there the operator has correctly entered the right side condition 252 of 18000 which would then be echoed in the left side of the screen of Fig. 5H upon depressing the enter key, thereby completing the query statement. The data output table provided by a database manager upon execution of this completed query is shown in Fig. 5I.

In the specification of a complex first component of a row clause wherein a summary function is defined (as was the case in the preceding example with reference to Fig. 5A-H), the argument of the summary function may desirably be not simply a column name but rather a complex expression. As an illustration of this, again with reference to the Staff table of Fig. 1, it may be desirable to retrieve from such a table a subset of data which is a list of employee names and their departments, but to limit such reported names and departments to only those wherein the sum of the salary and commission for every employee in the department is between \$20,000 and \$40,000. This is to be contrasted with the prior example wherein the minimum and maximum salaries reported in the data output table for each department were limited to only those wherein the respective department had an average salary less than \$18,000. In other words the distinction in the instant example is that the argument "salary + commission" to the summary function "sum" in this example is itself an expression whereas the argument "salary" of the summary function "avg" in the prior example was simply a column name selectable from the column names listed in the Summary

Function Items window prompt 243 of Fig. 5F.

In the present example under consideration, the correlative complex language SQL query select statement would be of the form

"SELECT NAME, DEPT FROM Q.STAFF GROUP BY NAME, DEPT HAVING SUM (SALARY + COMM)
5 BETWEEN 20,000 AND 40,000".

The window prompts and corresponding appropriate operator responses thereto to formulate an equivalent query statement in accordance with the present invention will now be discussed with reference to
10 Figs. 6A-F. As shown at the left-hand side of the prompt window of Fig. 6A, the Staff table and column names "NAME" and "DEPT" have already been selected from prior prompt windows in a manner hereinbefore described with respect to the preceding examples. When the prompt for definition of a row clause thus automatically appears as Rows condition prompt window 253, the operator is thus made aware that the first component of the row clause may be a column, expression or function.

15 Upon designating desire to specify this first condition as a Summary Function indicated at 254, the summary functions prompt window 255 appears as shown in Fig. 6B. The operator has selected as the desired summary function "SUM", 256, whereupon the Summary Function Items prompt window 257 appears in the screen depicted in Fig. 6C. In this screen the operator is made aware that the argument of the previously selected summary function may thus be a column selected from the column names list 258
20 or an expression. As indicated by the caret at 259, the operator has correctly responded to this prompt by indicating that it is desired to specify a complex first component of the row condition, namely an expression for the left side row condition.

Upon such indication, the expression prompt window 260 automatically appears in Fig. 6D requesting a keyed input from the operator of an expression for the summary function argument. As shown at 261, the
25 operator has thus appropriately keyed in the proper argument "SALARY + COMM". Upon completing inputting of the summary function argument, the second component of the row clause is automatically prompted for by means of the Comparison Operator's prompt window 262 automatically appearing in the prompt screen of Fig. 6E. In a manner consistent with the prior examples, the operator accordingly selects the appropriate verb "Is" 263 and comparison "Between" 264, thereby automatically resulting in appearance of the "Between" window prompt 265 (Fig. 6F). This prompts for appropriate response to define the
30 third component of the row clause thereby completing the row clause and the entire query statement. As shown in Fig. 6F, the operator, in response to being advised to input two items required by the "Between" comparison operator and further being advised that such items may be comprised of values, column names, or expressions, the operator thus properly keys in the right side values 20000 and 40000 as shown
35 by reference numeral 266, thereby completing the entire query statement. Upon execution of this query statement on the relational database, the resultant data output such as that shown in the form of the output table Fig 6G is produced.

With reference now to Figs. 8A-D, these figures illustrate in flow chart form the user interface software of the present invention which prompts the user to specify the clauses of a complex query statement
40 requiring minimal syntax and semantic knowledge from the user of the particular complex query language.

In Fig. 8A, the program starts at step 100 and proceeds to step 101 to begin the dialogue for the task, i.e., to begin receiving the keyed-in user input information to determine the table names for the query to be defined. Step 101 displays the prompt for selection of the tables to query. If the user enters more than one table, the user is automatically prompted to join the table as shown in Figure 2 block 181 and steps 107-
45 108 of Fig. 8A. The actual prompting of the Join condition is not discussed here in detail since neither it nor selection of multiple tables are requirements of this invention. In step 102, the user selects a table name and presses Enter to denote that the table name has been entered. Step 103 verifies the syntax of the table name is correct. If the syntax is not correct, an error message is displayed as shown in step 104 and the user is returned to step 101 to correct the table name. If the table name is syntactically correct, the
50 application determines in step 105 if the table exists in the database. Non-existent table names cause an error message to be displayed as shown in step 106 and the user is returned to step 101 to correct the table name. If the table name is semantically correct, the application determines in step 107 if more than one table was selected.

When more than one table is selected, the user is automatically prompted to join the tables as shown in
55 step 108. Otherwise, if only a single table is selected, the user is prompted for the next step in the sequence of steps as shown in step 109. Pressing the ESCAPE key at this time causes the prompting sequence to be terminated. If the user selects Tables as the next step, the user is taken through Tables prompting as shown in step 110 to define the Tables clause portion of the query statement. If the user

selects Columns as the next step, the user is similarly taken through Columns prompting as shown in step 111. By selection of Rows as the next step, the user is taken through Rows prompting as shown in step 112 and further detailed in Figs. 7, 8B, and 8C. If the user selects Sort as the next step, the user is taken through Sort prompting as shown in step 113. Finally, if the user selects Duplicates as the next step, the user is taken through Duplicates prompting as shown in step 114.

Rows prompting has been disclosed in great detail with reference to Figs. 8A-D because many features of the invention may be seen therefrom. However it will be appreciated that similar detailed prompting steps would be provided for specifying tables, columns, sort, and duplicates clauses shown only generally at 110-114.

After the tables prompting sequence 110 is completed, the program will loop back to connector A at 100. However after completion of the prompting for columns 111, sort 113, and duplicates 114, the program will loop back to connector Z at 170. This is also true for rows prompting sequence 112. However because the flow diagram for this latter sequence has been expanded in Fig. 8B connector C at 115 (Fig. 8A) is shown connecting to the expanded flow diagram at Fig. 8B rather than connector Z. However connector Z is shown at the bottom of Fig. 8B indicating looping back to connector Z at 170 in Fig. 8A after completion of the more detailed description of the row prompt sequence.

When the user selects Rows prompting as shown in step 112, processing continues through connector C (reference numeral 115) to step 120 in Figure 8B. If this is the first row condition clause to be specified, the user is prompted with the Row conditions window as shown in step 122. If this is a complex predicate and the user has specified a row condition already, the user is prompted for a row connector first as shown in step 121. In other words, a row clause may actually be comprised of a joinder of two or more such clauses as, for example, in the illustration "If salary = \$20,000 AND/OR COMMISSION = \$500". From the Row conditions window as shown in step 122, the user can select as shown in Fig. 8B a column name, step 123, an expression, step 125, or a summary function, step 132 as the left side or first component of a row condition or clause. Steps defining this first component are from 123-140.

If the user selects a column name from step 123, the column name is echoed to the display followed by an ellipsis as shown in step 124. If the user elects to define an expression at step 125, the user is prompted to enter an expression as shown in 126. When the user completes the expression and presses enter, the expression is validated for correct syntax as in step 127. If the user has entered a syntax error, a message is displayed as shown in step 128 and the user is returned to the expression prompting step 126 to correct the expression. However, if the expression is syntactically correct, the expression is validated for semantics as in step 129. If the user entered a semantic error (e.g., entered a column name that is not part of the table previously selected), a message is displayed as shown in step 130 and the user is returned to the expression prompting step 126 to correct the expression. Upon entry of a semantically correct expression, the expression is echoed to the display followed by an ellipsis from step 131.

If the user selects a summary function at step 132, the user is first prompted to select one of the summary functions as in step 133. The user may select one or more summary functions at this time. If more than one summary function is selected, the same argument is used for each summary function selected. When the user selects the summary function(s), the summary function(s) are echoed to the display followed by a left parenthesis and an ellipsis, step 134. The user is then prompted for the argument of the summary function, step 135. The user may select a column name as the argument, step 136, or define an expression as in step 137. If the user selects a column name, the column name is echoed to the display followed by a closing right parenthesis and then followed by an ellipsis. If the user selects an expression, the user is prompted for the expression as shown in step 139. At this time, the expression is syntactically and semantically checked as in steps 127-130. If errors occur, error messages are displayed as in steps 128 and 130 and the user is returned to step 139. If the expression is valid however, the expression is echoed to the display by step 140 followed by a closing right parenthesis followed by an ellipsis.

This completes the processing of the left side or first component of a row condition and processing now continues through connector X (reference numeral 141) to step 150 (Fig. 8C) where the left side of the row condition is checked to determine its data type. The valid indicators for the operators are set depending on the data type of the left side of the expression as shown in step 150. When the valid operators are determined, the comparison operators are displayed as shown in step 151. When a valid operator is selected, the selected operator is echoed to the display preceded by the verb IS or IS NOT depending on what the user selected from step 151A. The operator is then prompted for proper responses to specify the second component of the row clause. If the operator Equal to was selected, the Equal to window is displayed as shown in step 152. Selection of the Greater Than or Less Than operator causes the Less Than or Greater Than window to be displayed as shown in step 153. If the Starting With, Ending With, or

Containing operator was selected, the Starting With, Ending With, or Containing window is displayed, step 154. When the user completes the entry of the third component of the row condition as in 151B, the input will be eventually validated for correct syntax and semantics as shown in step 157. If an error occurred as shown in step 156, an error message is displayed and the user is returned to the preceding window as shown in step 152, 153, or 154. If the input is correct, the right side or third component of the row condition prompted for is entered at 151B (a value, column or expression), and is then echoed to the display as in step 158. By selecting the NULL operator, step 155, processing also proceeds to step 158. Upon completion of the right side, the user is returned to step 109 in Fig. 8A through connector Z (170) of Fig. 8A for continued prompts and responses to define the next clause of the query.

Referring now to Fig. 8A when the user escapes or terminates from the prompting sequence at 360, the application waits for more user input as shown in step 300 of Fig. 8D. If the user input that follows is a RUN command as in step 302, the application translates the stored query information, i.e., the table names, column, row conditions, etc., into an equivalent SQL SELECT statement that can be processed by the database as shown in step 303. When the database processes the query, the results are returned from the database to the user as shown in step 304. The results may be shown to the user in the form of a Report with column headings above the columns of information returned 307. The application then loops back to 360 awaiting more user input. If the user input is not a RUN command, the application determines if an EXIT command has been issued as in step 306. If the command is Exit, the application is terminated. If another command has been issued, the command is processed as in step 305 and the application waits for further user input at 300.

Fig. 11 is a simplified schematic diagram of the computerised system of the present invention for generating search commands in the manner hereinbefore described. First, a processor 350 is provided which may be in the form of any number of CPU's in micro, mini, and mainframe computers currently available such as those contained in the conventional and familiar personal computer or "PC". A memory 352 is further provided. This memory stores the database manager program which includes the query prompting program hereinbefore described in greater with reference to Figs. 8A-8D. Additionally, memory 352 stores relational data such as that of Fig. 1 to be queried in accordance with the present invention as well as operator inputs keyed in by means of a keyboard 351. This user input data may include additional data inputs to the relational data stored in the memory 352, updating and editing commands therefor, as well as the various database search commands or queries which may be constructed in accordance with the invention as described herein.

A display screen 353 is further provided which may be of any design appropriate for the display of prompt screens such as those depicted in Figs. 3A-3E. The purpose of this display 353 is to enable the operator to view the various prompting screens generated by the present invention including the various instructions and valid response choices contained therein, as well as to provide a convenient means for viewing the various operator inputs keyed into the keyboard 351. Yet an additional purpose of the display 353 is to provide a visual indication of the data output tables such as that of Fig. 5I resulting from execution of the query formulated by the user in response to the prompted queries of the invention. Finally, with reference to Fig. 11 a conventional connection 354 provides electrical innerconnection between the processor 350, memory 352, keyboard 351, and display 353 so as to facilitate intercommunication between these components as required. The connection 354 is shown in a general schematic form. However it will be readily appreciated by those of skill in the art that portions of this innerconnection 354 may take the form of conventional data, control, and address lines, buses and the like.

With reference to Fig. 10, in order to better appreciate the benefits of the present invention in simplifying formulation of queries by eliminating need for syntax and semantic knowledge, direct creation of complex language queries will now be discussed in greater detail using SQL as but one representative example. This is not intended to be an exhaustive discussion but rather only serves to illustrate the complexity involved in formulating directly such queries in the query language.

For the purposes of this invention, Figure 10 defines the elements of a complex statement, SELECT, as defined in the Structure Query Language (SQL). This statement defines the clauses that a user would need to know in order to construct a syntactically and semantically correct SQL SELECT statement. The SELECT statement is used to interface with a database manager for the purposes of retrieving specific data from the database.

Reference is now made to Figure 10 which shows the syntax for the SELECT statement. The SELECT statement is used to return a table of values to the end user. The SELECT statement may consists of up to 6 clauses each starting with the keyword or clause SELECT 10, FROM 11, WHERE 12, GROUP BY 13, HAVING 14, and ORDER BY 15.

With reference to Fig. 9, in the discussion which follows SQL clause keywords 97 will be used each of

which has a correlative and analogous non-SQL clause keyword 98 previously discussed with reference to Fig. 2 and the preceding discussion of the invention.

For the purposes of describing the syntax, the following statement convention has been used. Brackets 16 are used to denote optional items. Braces 17 are used when it is necessary to group items that are required. When there is a choice of keywords, the items will be separated by a vertical line 18. Ellipses 19
5 are used to indicate that an item can be repeated. The clauses 10, 11, 12, 13, 14, and 15 must be specified in the order shown if present.

The following describes each clause of the SELECT statement. As hereinbefore noted this description is not an exhaustive description of the syntax and semantics. The intent is to provide an insight into the complexity of the statement and the amount of syntax and semantic knowledge required by the end user without benefit of the invention.

The SELECT clause describes the fields to be returned in the output table. The output table is the table of values returned to the end user upon execution of the SELECT statement by the database manager. There are two parts to this clause. The first part 20 and 21 specifies whether identical rows are to be combined, and the second part 22 and 23 describes each field of the output. ALL (20) specifies that one
15 output row is returned for each row of the final result table, without eliminating duplicate rows. DISTINCT 21 specifies that duplicate output rows are to be eliminated. The asterisk (*) 22 specifies that one field is returned for each column in the table definition specified by the FROM clause 11. This is a shorthand for writing out the list of column names. The select list 23 is a list that contain expressions. An expression
20 specifies a single output field and can be a constant 40, a column name 41, an arithmetic expression 42, or a column function 43. Expressions are values derived by combining one or more fields or functions with arithmetic operators (+, -, *, /) and parentheses. The result is always a single value. Character strings cannot be combined in expressions. An example of an expression is SALARY+COMM. If the expression contains column names, the values are substituted for the selected row in determining the output field value
25 for that row.

A constant 40 returns the same value for each row. A constant is a string of characters embedded in a statement string that represents a value. A constant may be a string constant or numeric constant. A string constant is enclosed in single quotes. Numeric constants include integer, decimal, and floating point numbers.

A column name 41 returns the value in the table column for the selected row. A column is a field that represents the value of one column of the current row of a table. An arithmetic expression 42 returns the value computed from the expression. A column function 43 returns a single value based on the function specified. The functions provided include AVG, COUNT, MIN, MAX, and SUM. The arguments of a function are enclosed in parentheses 50. The result of a function is derived by the application of the function to the
35 specified arguments. Column functions provide a capability to return a value computed over values of the output rows. Thus it logically returns only one row of output for the SELECT statement.

AVG 51 is the average values returned for each row. Expression 44 is any valid expression that returns a numeric value. COUNT 52 returns an integer containing the count value. Asterisk 45 returns a count of the number of rows in the output set, and DISTINCT column name 46 returns a count of the distinct values
40 found in the column. MIN 53 and MAX 54 returns the minimum or maximum value. Expressions 47 and 48 are any valid expression that returns a numeric value. SUM 55 returns algebraic sum of an expression. Expression 49 is a valid numeric expression. If column functions 43 are used in a select-list 23 and the SELECT statement does not contain a GROUP BY clause 13, all fields in the select-list 23 must be column functions 43 or other expressions that do not include column names.

The FROM clause 11 specifies the names of the tables and views from which to retrieve the data specified by the SELECT clause 10. A FROM clause may contain one or more table name. If more than one table name is specified, the tables are joined together to define a cartesian product of the input tables. This is an elementary form of join. The WHERE clause 12 of the SELECT statement is used to reduce the number of rows brought to the output to a meaningful set. For the purposes of this invention, only one table
50 SELECT statements will be discussed. A correlation name 25 may be specified and is an alternate name used for a table name. The name is used in place of the table name in qualifying a column name anywhere in the SELECT statement.

The WHERE clause 12 includes a single predicate 26 (simple or complex) used to determine which input table rows generate output data. A simple predicate is a test between two expressions 62 and 60 and
55 an operator 61. The format for the first six operators 63, 64, 65, 66, 67, and 68 is obvious. The BETWEEN 69 predicate is satisfied if the value of expression1 70 is equal to or greater than expression2 72 and equal to or less than expression3 73. The NOT keyword 71 yields a value of true if expression1 70 is less than expression2 72 or greater than expression3 73.

The IN 74 predicate is true if expression1 75 matches expression2 77, or if the NOT keyword 76 is used, does not match expression2 77. This is equivalent to a simple predicate of the form: expression1 = expression2. The IN 74 predicate is true if expression1 78 matches any of the values in the list 80, or if the NOT keyword 79 is used, does not match any value in the list 80. The values in the list can be constants.

5 The IS NULL 81 predicate is true if the value of the column name 82 is null or, if the NOT keyword 83 is used, if it is not null. The LIKE 84 predicate is true if the value of the column name 85 matches a pattern specified in the expression 87, or if the NOT keyword 86 is used, does not match the pattern specified in the expression 87. The value of the column name 85 must be a character string. The expression 87 can be a character string. The pattern in a character string can optionally include special characters underscore and percent. The underscore character means to allow any character to fill the position in the string. The
10 percent means to allow any number of characters to fill that position in the string. For example, ABC could produce values 12ABC OR XZABC. %ABC could produce ABC, 12ABC, OR 123456ABC. __ABC%D% could produce 1ABCD, 2ABCXXXX, or 3ABCDXXXX.

Complex predicates combine several simple predicates together to yield a single truth value of true or
15 false. Simple predicates can be combined using the AND and OR operators. A WHERE clause 12 is not required. The GROUP BY clause 13 is used in conjunction with column functions 43 to deliver summary rows. Without this clause, a select-list 23 with column functions returns a single row for the entire SELECT statement. With this clause, it returns a row having the column function values for each distinct value of the fields in the GROUP BY clause. Column name 27 defines the name of a column that exists in the input
20 table. This clause allows the column names used in the GROUP BY clause to be used in the select-list 23. They are not required to be in the select-list.

The HAVING clause 14 is used to qualify summary rows for output. A summary row results when column functions 43 are used in the select-list 23. If no GROUP BY clause 13 is used, there is one summary row; otherwise, there are multiples to be tested against the HAVING clause 14. A predicate 28 is
25 a standard complex predicate where the expressions can contain only constants 40, column names 27 used in the GROUP BY clause 13, or column functions 43 used in the select-list 23. A statement may contain both a WHERE clause 12 and a HAVING clause 14. The WHERE clause determines which rows of the input table to use in generating a summary row. The HAVING clause tests the summary row for selection as an output row.

30 The ORDER BY clause 15 is used to order the output rows. If not used, the order of the rows returned is indeterminate. If more than one column name 29 is specified, the rows are ordered first by the values of the first identified column, then by the values of the second identified column, and so on. Column name 29 is the name of a column as used in the select-list. The name must be used in the select-list as a single item, not part of an expression. ASC 30 indicates ascending order for this field. DESC 31 indicates
35 descending order for this field.

While the invention has been shown and described with reference to particular embodiments thereof, it will be understood by those skilled in the art that the foregoing and other changes in form and details may be made therein without departing from the spirit and scope of the invention.

40

Claims

1. A method of operating a computer system for generating a search command clause for accessing a database comprising the steps of:

45

generating and displaying a first prompt relating to a first component of the clause;
registering a response to said first prompt;
generating and displaying a second prompt relating to a second component of the clause in response to the registered response to said first prompt; and
50 registering a response to said second prompt,

wherein said registered responses comprise required components of the search command clause.

2. A method according to claim 1 wherein said search command clause has first, second and third components, and including the further steps of:

55

generating and displaying a third prompt for said third component of the clause; and
registering a response to the third prompt, said response comprising said third component of the search command.

3. A method according to claim 1 or claim 2, in which said first component of the search clause is a complex function.

4. A method according to claim 3, in which said complex function is a summary function or an expression.

5 5. A method according to any of the previous claims, in which each pair of steps of generating and displaying and registering comprises the steps of:

generating and displaying a prompt window with a plurality of response choices; and
registering selection of one or more of the response choices.

10 6. A method according to any one of the previous claims, wherein said second component is a comparison operator, said second prompt includes response choices for defining said comparison operator, and said second prompt is automatically displayed upon registration of the response to the first prompt.

7. A data processing system for database operation comprising means for generating and displaying a first prompt relating to a first component of a search command clause for the database, means for
15 registering a response to said first prompt, means for generating and displaying a second prompt relating to a second component of the clause and means for registering a response to said second prompt, wherein said registered responses comprise required components of the search command clause.

8. A system according to claim 7 wherein said search command clause has first, second and third components, and including means for generating and displaying a third prompt for said third component of
20 the clause, and means for registering a response to the third prompt, said response comprising said third component of the search command.

9. A system according to claim 7 or claim 8, in which said first component of the search clause is a complex function.

10. A system according to claim 9, in which said complex function is a summary function or an
25 expression.

11. A system according to any one of claims 7 to 10, in which each means for generating and displaying and for registering comprises means for generating and displaying a prompt window with a plurality of response choices, and means for registering selection of one or more of the response choices.

12. A system according to any one of claims 7 to 11, wherein said second component is a comparison
30 operator, said second prompt includes response choices for defining said comparison operator, and said second prompt is automatically displayed upon registration of the response to the first prompt.

35

40

45

50

55

STAFF

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
10	SANDERS	20	MGR	7	18357.50	-
20	PERNAL	20	SALES	8	18171.25	612.45
30	MARENGHI	38	MGR	5	17506.75	-
40	O'BRIEN	38	SALES	6	18006.00	846.55
50	HANES	15	MGR	10	20659.00	-
60	QUIGLEY	38	SALES	-	16808.30	650.25
70	ROTHMAN	15	SALES	7	15502.83	1152.00
80	JAMES	20	CLERK	-	13504.60	128.20
90	KOONITZ	42	SALES	6	18001.75	1386.70
100	PLOTZ	42	MGR	7	18352.80	-
110	NGAN	15	CLERK	5	12508.20	206.60
120	NAUGHTON	38	CLERK	-	12954.75	180.00
130	YAMAGUCHI	42	CLERK	6	10505.90	75.60
140	FRAYE	51	MGR	6	21150.00	-
150	WILLIAMS	51	SALES	6	19456.50	637.65
160	MOLINARE	10	MGR	7	22959.20	-
170	KERMISCH	15	CLERK	4	12258.50	110.10
180	ABRAHAM	38	CLERK	3	12009.75	236.50
190	SNEIDER	20	CLERK	8	14252.75	126.50
200	SCOUTTEN	42	CLERK	-	11508.60	84.20
210	LU	10	MGR	10	20010.00	-
220	SMITH	51	SALES	7	17654.50	992.80
230	LUNDQIST	51	CLERK	3	13369.80	189.65
240	DANIELS	10	MGR	5	19260.25	-
250	WHEELER	51	CLERK	6	14460.00	513.30
260	JONES	10	MGR	12	21234.00	-
270	LEA	66	MGR	9	18555.50	-
280	WILSON	66	SALES	9	18674.50	811.50
290	QUILL	84	MGR	10	19818.00	-
300	DAVIS	84	SALES	5	15454.50	806.10
310	GRAHAM	66	SALES	13	21000.00	200.30
320	GONZALES	66	SALES	4	16858.20	844.00
330	BURKE	66	CLERK	1	10988.00	55.50
340	EDWARDS	84	SALES	7	17844.00	1285.00
350	GAFNEY	84	CLERK	5	13030.50	188.00

FIG. 1

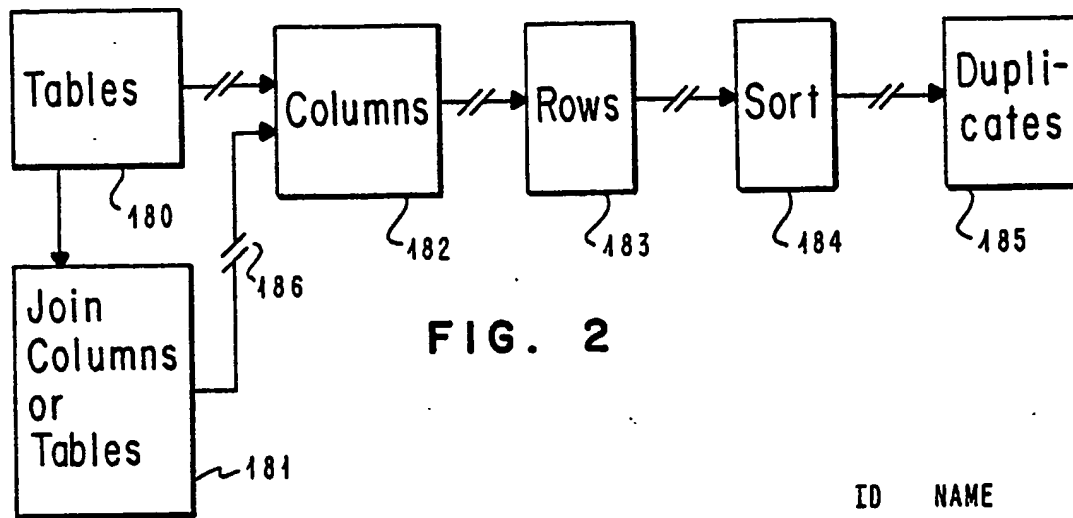


FIG. 2

ID	NAME
150	WILLIAMS
250	WHEELER
280	WILSON

FIG. 3F

ID	NAME	SALARY
20	PERNAL	18171.25
40	O'BRIEN	18006.00
90	KOONITZ	18001.75
150	WILLIAMS	19456.50
220	SMITH	17654.50
280	WILSON	18674.50
310	GRAHAM	21000.00
340	EDWARDS	17844.00

FIG. 4E

DEPT	MIN(SALARY)	MAX(SALARY)
15	12258.50	20659.80
20	13504.60	18357.50
38	12009.75	18006.00
42	10505.90	18352.80
51	13369.80	21150.00
66	10988.00	21000.00
84	13030.50	19818.00

FIG. 5I

NAME	DEPT
GRAHAM	66
WILLIAMS	51

FIG. 6G

Actions	Edit	Specify	Display	Exit	F1=Help																						
<p>Prompted Query "----NEW----" (Scroll)</p>																											
<p>Select Edit above to revise a</p>																											
<p>Tables:</p> <p>Q. STAFF</p>																											
<p>Columns:</p> <p>> ...</p>																											
<div> <div>Columns</div> <div> <p>Select one or more columns, or select either expression or function; then press Enter.</p> <table border="1"> <thead> <tr> <th>Column Names</th> <th>(scroll)</th> </tr> </thead> <tbody> <tr> <td>STAFF - all</td> <td></td> </tr> <tr> <td>ID</td> <td></td> </tr> <tr> <td>NAME</td> <td></td> </tr> <tr> <td>DEPT</td> <td></td> </tr> <tr> <td>JOB</td> <td></td> </tr> <tr> <td>YEARS</td> <td></td> </tr> <tr> <td>SALARY</td> <td></td> </tr> <tr> <td>COMM</td> <td></td> </tr> <tr> <td>Expression (A+B) ...</td> <td></td> </tr> <tr> <td>Summary functions (SUM, etc.) ...</td> <td></td> </tr> </tbody> </table> </div> </div>						Column Names	(scroll)	STAFF - all		ID		NAME		DEPT		JOB		YEARS		SALARY		COMM		Expression (A+B) ...		Summary functions (SUM, etc.) ...	
Column Names	(scroll)																										
STAFF - all																											
ID																											
NAME																											
DEPT																											
JOB																											
YEARS																											
SALARY																											
COMM																											
Expression (A+B) ...																											
Summary functions (SUM, etc.) ...																											
<p>Enter Esc=Cancel F1=Help</p>																											

FIG. 3B

Actions	Edit	Specify	Display	Exit	F1=Help
---------	------	---------	---------	------	---------

Prompted Query "-----NEW-----" (Scroll) 196

Select Edit above to revise a

Tables:

- ☐ Q. STAFF

Columns:

- ☐ ID 204
- ☐ NAME

Row Conditions:

> If...

Rows

Select a column, an expression, or function; then press Enter.

Column Names	(Scroll)
STAFF	
<input type="checkbox"/> ID	
>	
<input type="checkbox"/> NAME	
<input type="checkbox"/> DEPT	
<input type="checkbox"/> JOB	
<input type="checkbox"/> YEARS	
<input type="checkbox"/> SALARY	
<input type="checkbox"/> COMM	
<input type="checkbox"/> YEARS	
<input type="checkbox"/> Expression (A+B, etc.)	
<input type="checkbox"/> Summary function (SUM, etc.)...	

Esc=Cancel
F1=Help

FIG. 3C

Actions	Edit	Specify	Display	Exit	F1=Help
Prompted Query "----NEW----" (Scroll)					
Select Edit above to revise a					
Tables:					
<input type="checkbox"/> Q. STAFF					
Columns:					
<input type="checkbox"/> ID					
<input type="checkbox"/> NAME					
Row Conditions:					
> If NAME ...					
<div style="border: 1px solid black; padding: 10px;"> <div style="display: flex; justify-content: space-between;"> Verb > Is </div> <div style="display: flex; justify-content: space-between;"> Comparison > Is not </div> <div style="display: flex; justify-content: space-between;"> > Equal to . . . </div> <div style="display: flex; justify-content: space-between;"> > Less than . . . </div> <div style="display: flex; justify-content: space-between;"> > Greater than . . . </div> <div style="display: flex; justify-content: space-between;"> > Between </div> <div style="display: flex; justify-content: space-between;"> > Starting with . . . </div> <div style="display: flex; justify-content: space-between;"> > Ending with . . . </div> <div style="display: flex; justify-content: space-between;"> > Containing . . . </div> <div style="display: flex; justify-content: space-between;"> > NULL </div> </div>					
Enter Esc=Cancel F1=Help					

FIG. 3D

Actions	Edit	Specify	Display	Exit	F1=Help
Prompted Query "----NEW----"					(Scroll)
Select Edit above to revise a					
Tables:					
Q. STAFF					
Columns:					
ID					
NAME					
Row Conditions:					
> If NAME Starts with...					
Starting With					
Type one or more values; then press Enter.					
Starting with .. [W					
Or					
Or					
Or					
Or					
Or					
Enter Esc=Cancel F1=Help					
NULL					
Enter Esc=Cancel F1=Help					

FIG. 3E

Actions	Edit	Specify	Display	Exit	F4=Help
---------	------	---------	---------	------	---------

Prompted Query. "----NEW----" (Scroll)

Select Edit above to revise a

Tables: Q. STAFF 220

Columns:

ID	NAME	SALARY
----	------	--------

Row Conditions:

> If...

Rows ~ 224

Select a column, an expression, or function; then press Enter.

Column Names (Scroll)	(Scroll)
STAFF	
ID	
NAME	
DEPT	
JOB	
YEARS	
SALARY	
COMM	

> Expression (A+B, etc.)...

Summary function (SUM, etc.)...

Esc=Cancel F4=Help

FIG. 4A

Actions	Edit	Specify	Display	Exit	F1=Help
---------	------	---------	---------	------	---------

Prompted Query "-----NEW-----" (Scroll)

Select Edit above to revise a

Tables:

- Q. STAFF

Columns:

- ID
- NAME
- SALARY

Row Conditions:

Rows

Select a column, an expression, or function; then press Enter.

Column Names	(Scroll)
STAFF	
ID	
NAME	
DEPT	

Expression

Type an expression. You can use the following arithmetic operators:
add (+), subtract (-), multiply (*), and divide (/); then press Enter.

[SALARY + COMM

>

Enter Esc=Cancel F1=Help F4=List

FIG. 4B

Actions	Edit	Specify	Display	Exit	F1=Help
---------	------	---------	---------	------	---------

Prompted Query "----NEW----" (Scroll)

Select Edit above to revise a

Tables:

- Q. STAFF

Columns:

- ID
- NAME
- SALARY

Row Conditions:

- > If SALARY+COMM...

Rows 227

Comparison Operators

Select the verb and a comparison; then press Enter.

Verb	>	Is
Comparison		Is not
		Equal to
		Less than
		Greater than
		Between
		Starting with
		Ending with
		Containing
		NULL

Enter Esc=Cancel F1=Help

FIG. 4C

Actions	Edit	Specify	Display	Exit	:	F1=Help
Prompted Query " -- -- NEW -- -- " (Scroll)						
Select Edit above to revise a						
Tables:						
Q. STAFF						
Columns:						
ID						
NAME						
SALARY						
Row Conditions:						
> If SALARY+COMM Is Greater						

Rows

Comparison Operators

Greater Than²³²
Type a value, column name, or expression;
then press Enter.
18000²³³ >

Enter Esc=Cancel F1=Help F4=List

Between...
Starting with...
Ending with...
Containing...
NULL

Enter Esc=Cancel F1=Help

FIG. 4D

Actions	Edit	Specify	Display	Exit	F1=Help
---------	------	---------	---------	------	---------

Prompted Query "----NEW----" (Scroll)

Select Edit above to revise a

Tables:

- Q. STAFF

Columns:

- DEPT
- > ...

Columns ~232

Select one or more columns, or select either expression or function; then press Enter.

Column Names	(scroll)
<input checked="" type="checkbox"/> STAFF - all	
<input type="checkbox"/> ID	
<input type="checkbox"/> NAME	
<input type="checkbox"/> DEPT	
<input type="checkbox"/> JOB	
<input type="checkbox"/> YEARS	
<input type="checkbox"/> SALARY	
<input type="checkbox"/> COMM	

☒ Expression (A+B)...

> Summary functions (SUM, etc.)...

Enter Esc=Cancel F1=Help

FIG. 5A

Actions	Edit	Specify	Display	Exit	F1=Help
---------	------	---------	---------	------	---------

Select Edit above to revise a

Tables:

- Q. STAFF

Columns:

- DEPT
- > ...

Prompted Query "----NEW----" (Scroll)

Columns ²³⁴

Summary Functions

Select one or more items;
then press Enter.

- 1. Count of (COUNT)
- 2. Average of (AVG)...
- 3. Sum of (SUM)...
- 4. Minimum of (MIN)...
- 5. Maximum of (MAX)...

Enter Esc=Cancel F1=Help

select either
ss Enter.

(scroll).

Expression (A+B)...

> Summary functions (SUM, etc.) ...

Enter Esc=Cancel F1=Help

FIG. 5B

Actions

Edit

Specify

Display

Exit

F1=Help

Prompted Query "----NEW----" (Scroll)

Select Edit above to revise a

Tables:

- Q. STAFF

Columns:

- DEPT
- MIN(...)
- MAX(...)

Summary Function Items

Select a column, or an expression; then press Enter.

Column Names (scroll)

STAFF

ID

NAME

DEPT

JOB

YEARS

SALARY

COMM

Expression (A+B, etc.)...

Esc = Cancel F1 = Help

FIG. 5C

ActionsEditSpecifyDisplayExit

F1=Help

Prompted Query "----NEW ----" (Scroll)

Select Edit above to revise a

Tables:

- Q. STAFF

Columns:

- DEPT
- MIN(SALARY)
- MAX(SALARY)

Row Conditions:

- If ...

239

Rows

Select a column, an expression, or function; then press Enter.

Column Names (Scroll)

MIN(SALARY)

MAX(SALARY)

STAFF

ID

NAME

DEPT

JOB

YEARS

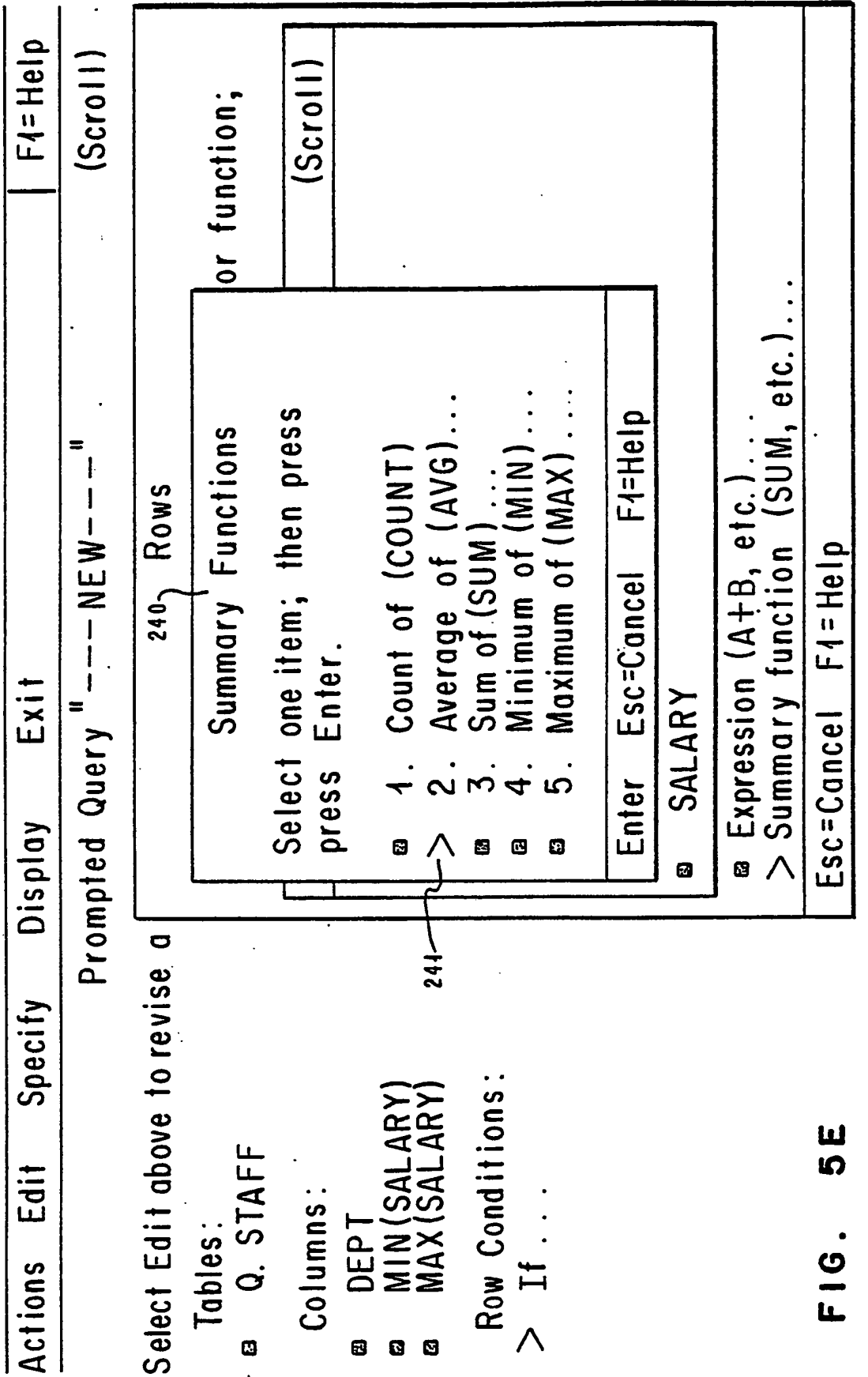
SALARY

Expression (A+B, etc.)...

Summary function (SUM, etc.)...

Esc=Cancel F1=Help

FIG. 5D



Actions	Edit	Specify	Display	Exit	F1=Help																				
<div style="display: flex; justify-content: space-between; align-items: center;"> <div> <p>Select Edit above to revise a</p> <p>Tables:</p> <ul style="list-style-type: none"> <input type="checkbox"/> Q. STAFF <p>Columns:</p> <ul style="list-style-type: none"> <input type="checkbox"/> DEPT <input type="checkbox"/> MIN(SALARY) <input type="checkbox"/> MAX(SALARY) <p>Row Conditions:</p> <p>> If AVG(...) 242</p> </div> <div> <p>Prompted Query " --- NEW --- " 243</p> </div> <div> <p>(Scroll)</p> </div> </div>																									
<div style="border: 1px solid black; padding: 10px;"> <div style="display: flex; justify-content: space-between;"> <div> <p>Summary Function Items</p> <p>Select a column, or an expression; then press Enter.</p> </div> <div> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 50%;">Column Names</th> <th style="width: 50%;">(scroll)</th> </tr> </thead> <tbody> <tr><td><input type="checkbox"/> STAFF</td><td></td></tr> <tr><td><input type="checkbox"/> ID</td><td></td></tr> <tr><td><input type="checkbox"/> NAME</td><td></td></tr> <tr><td><input type="checkbox"/> DEPT</td><td></td></tr> <tr><td><input type="checkbox"/> JOB</td><td></td></tr> <tr><td><input type="checkbox"/> YEARS</td><td></td></tr> <tr><td><input type="checkbox"/> ></td><td></td></tr> <tr><td><input type="checkbox"/> SALARY</td><td>244</td></tr> <tr><td><input type="checkbox"/> COMM</td><td></td></tr> </tbody> </table> </div> </div> <div style="border-top: 1px solid black; padding-top: 5px;"> <input type="checkbox"/> Expression (A+B, etc.) ... </div> <div style="display: flex; justify-content: space-between; border-top: 1px solid black; padding-top: 5px;"> Esc = Cancel F1 = Help </div> </div>						Column Names	(scroll)	<input type="checkbox"/> STAFF		<input type="checkbox"/> ID		<input type="checkbox"/> NAME		<input type="checkbox"/> DEPT		<input type="checkbox"/> JOB		<input type="checkbox"/> YEARS		<input type="checkbox"/> >		<input type="checkbox"/> SALARY	244	<input type="checkbox"/> COMM	
Column Names	(scroll)																								
<input type="checkbox"/> STAFF																									
<input type="checkbox"/> ID																									
<input type="checkbox"/> NAME																									
<input type="checkbox"/> DEPT																									
<input type="checkbox"/> JOB																									
<input type="checkbox"/> YEARS																									
<input type="checkbox"/> >																									
<input type="checkbox"/> SALARY	244																								
<input type="checkbox"/> COMM																									

FIG. 5F

Prompted Query "----NEW----" (Scroll)

Select Edit above to revise a

Tables:

Q. STAFF

Columns:

DEPT

MIN(SALARY)

MAX(SALARY)

Row Conditions:

> If AVG(SALARY)...

245

ROWS - 246

Comparison Operators

Select the verb and a comparison; then press Enter.

Verb

 \wedge

— 248 —

is not

Equal

Less

Great

Between

Starti

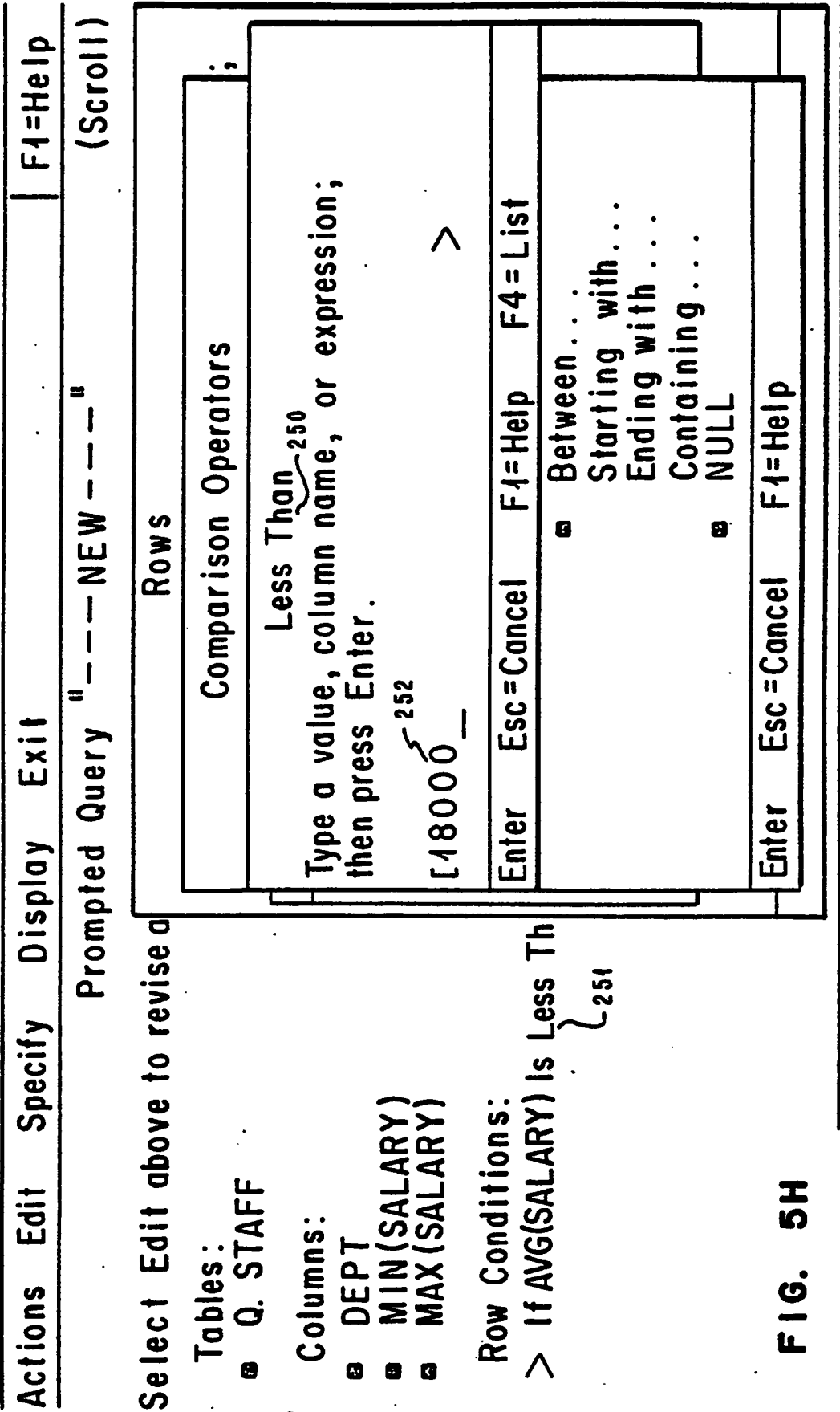
Ending

Contents

77N

Enter Esc=Cancel F1=Help

FIG. 5C



SELECT STATEMENT:
 SELECT NAME, DEPT
 FROM Q. STAFF
 GROUP BY NAME, DEPT
 HAVING SUM(SALARY+COMM) BETWEEN 20000 AND 40000

Actions	Edit	Specify	Display	Exit	F1=Help																				
Prompted Query "----NEW----" (Scroll)																									
Rows 253 Select a column, an expression, or function; then press Enter.																									
<table border="1"> <thead> <tr> <th>Column Names</th> <th>(Scroll)</th> </tr> </thead> <tbody> <tr><td>STAFF</td><td></td></tr> <tr><td>ID</td><td></td></tr> <tr><td>NAME</td><td></td></tr> <tr><td>DEPT</td><td></td></tr> <tr><td>JOB</td><td></td></tr> <tr><td>YEARS</td><td></td></tr> <tr><td>SALARY</td><td></td></tr> <tr><td>COMM</td><td></td></tr> <tr><td>YEARS</td><td></td></tr> </tbody> </table>						Column Names	(Scroll)	STAFF		ID		NAME		DEPT		JOB		YEARS		SALARY		COMM		YEARS	
Column Names	(Scroll)																								
STAFF																									
ID																									
NAME																									
DEPT																									
JOB																									
YEARS																									
SALARY																									
COMM																									
YEARS																									
Expression (A+B, etc.)... Summary function (SUM, etc.)...																									
Esc=Cancel F1=Help																									

Select Edit above to revise a

Tables:

Q. STAFF

Columns:

NAME
 DEPT

Row Conditions:

> If...

FIG. 6A

Actions Edit Specify Display Exit

Prompted Query "----NEW----" (Scroll)

Select Edit above to revise a

Tables:

- Q. STAFF

Columns:

- NAME
- DEPT

Row Conditions:

> If...

Summary Functions

- 1. Count of (COUNT)
- 2. Average of (AVG)...
- 3. Sum of (SUM)...
- 4. Minimum of (MIN)...
- 5. Maximum of (MAX)...

Enter Esc=Cancel F1=Help

Expression (A+B)...
Summary functions (SUM, etc.)...

Enter Esc=Cancel F1=Help

FIG. 6B

FIG. 6B

Actions	Edit	Specify	Display	Exit	F1-Help																		
Prompted Query "---- NEW----" ²⁵⁷ (Scroll)																							
Select Edit above to revise a																							
Tables: <ul style="list-style-type: none"> Q. STAFF 																							
Columns: <ul style="list-style-type: none"> NAME DEPT 																							
Row Conditions: <ul style="list-style-type: none"> > If SUM(... 																							
<div> <div>Summary Function Items</div> <div> Select a column, or an expression; then press Enter. </div> <table border="1"> <thead> <tr> <th>Column Names</th> <th>(scroll)</th> </tr> </thead> <tbody> <tr> <td>STAFF</td> <td></td> </tr> <tr> <td>ID</td> <td></td> </tr> <tr> <td>NAME</td> <td></td> </tr> <tr> <td>DEPT</td> <td></td> </tr> <tr> <td>JOB</td> <td></td> </tr> <tr> <td>YEARS</td> <td></td> </tr> <tr> <td>SALARY</td> <td></td> </tr> <tr> <td>COMM</td> <td></td> </tr> </tbody> </table> <div> > Expression (A+B, etc.)... </div> </div>						Column Names	(scroll)	STAFF		ID		NAME		DEPT		JOB		YEARS		SALARY		COMM	
Column Names	(scroll)																						
STAFF																							
ID																							
NAME																							
DEPT																							
JOB																							
YEARS																							
SALARY																							
COMM																							
En	Esc=Cancel	F1=Help																					

FIG. 6C

Actions Edit Specify Display Exit	F1=Help								
Prompted Query " ---NEW ---" (Scroll)									
Select Edit above to revise a Tables: Q. STAFF Columns: NAME DEPT Row Conditions:	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> Summary Function Items Select a column, or an expression; then press Enter. <table border="1" style="width: 100%; margin-top: 5px;"> <tr> <td style="width: 50%; text-align: center;">Column Names</td> <td style="width: 50%; text-align: center;">(scroll)</td> </tr> <tr> <td style="text-align: center;">STAFF</td> <td></td> </tr> <tr> <td style="text-align: center;">ID</td> <td></td> </tr> <tr> <td style="text-align: center;">NAME</td> <td></td> </tr> </table> </div> <div style="border: 1px solid black; padding: 5px;"> Expression Type an expression. You can use the following arithmetic operators: add(+), subtract (-), multiply (*), and divide (/); then press Enter. <div style="border: 1px solid black; padding: 5px; margin-top: 5px;"> [SALARY + COMM_ > </div> </div>	Column Names	(scroll)	STAFF		ID		NAME	
Column Names	(scroll)								
STAFF									
ID									
NAME									
Enter Esc=Cancel	F1=Help F4=List								

FIG. 6D

Actions	Edit	Specify	Display	Exit	F1=Help
---------	------	---------	---------	------	---------

Prompted Query "-----NEW-----" (Scroll)

Select Edit above to revise a

Tables:

- ☐ Q. STAFF

Columns:

- ☐ NAME
- ☐ DEPT

Row Conditions:

> If SUM(SALARY+COMM)...

Rows
262

Comparison Operators

Select the verb and a comparison; then press Enter.

Verb	>	Is	
Comparison	<input type="checkbox"/>	Is not	
	<input type="checkbox"/>	Equal to . . .	
	<input type="checkbox"/>	Less than . . .	
	<input type="checkbox"/>	Greater than . . .	
	<input type="checkbox"/>	Between	
	<input type="checkbox"/>	Starting with . . .	
	<input type="checkbox"/>	Ending with . . .	
	<input type="checkbox"/>	Containing . . .	
	<input type="checkbox"/>	NULL	

Enter
Esc=Cancel
F1=Help

FIG. 6E

Actions	Edit	Specify	Display	Exit	F1=Help
Prompted Query "-----NEW-----"					(Scroll)
Select Edit above to revise a					
Tables:					
<input type="checkbox"/> Q. STAFF					
Columns:					
<input type="checkbox"/> NAME <input type="checkbox"/> DEPT					
Row Conditions:					
> If SUM(SALARY+COMM)I					
Type two values, column names, or expressions; then press Enter.					
Between [20000] And [40000]					
Enter Esc=Cancel F1=Help F4=List					
Starting with... Ending with... Containing... NULL					
Enter Esc=Cancel F1=Help					

FIG. 6F

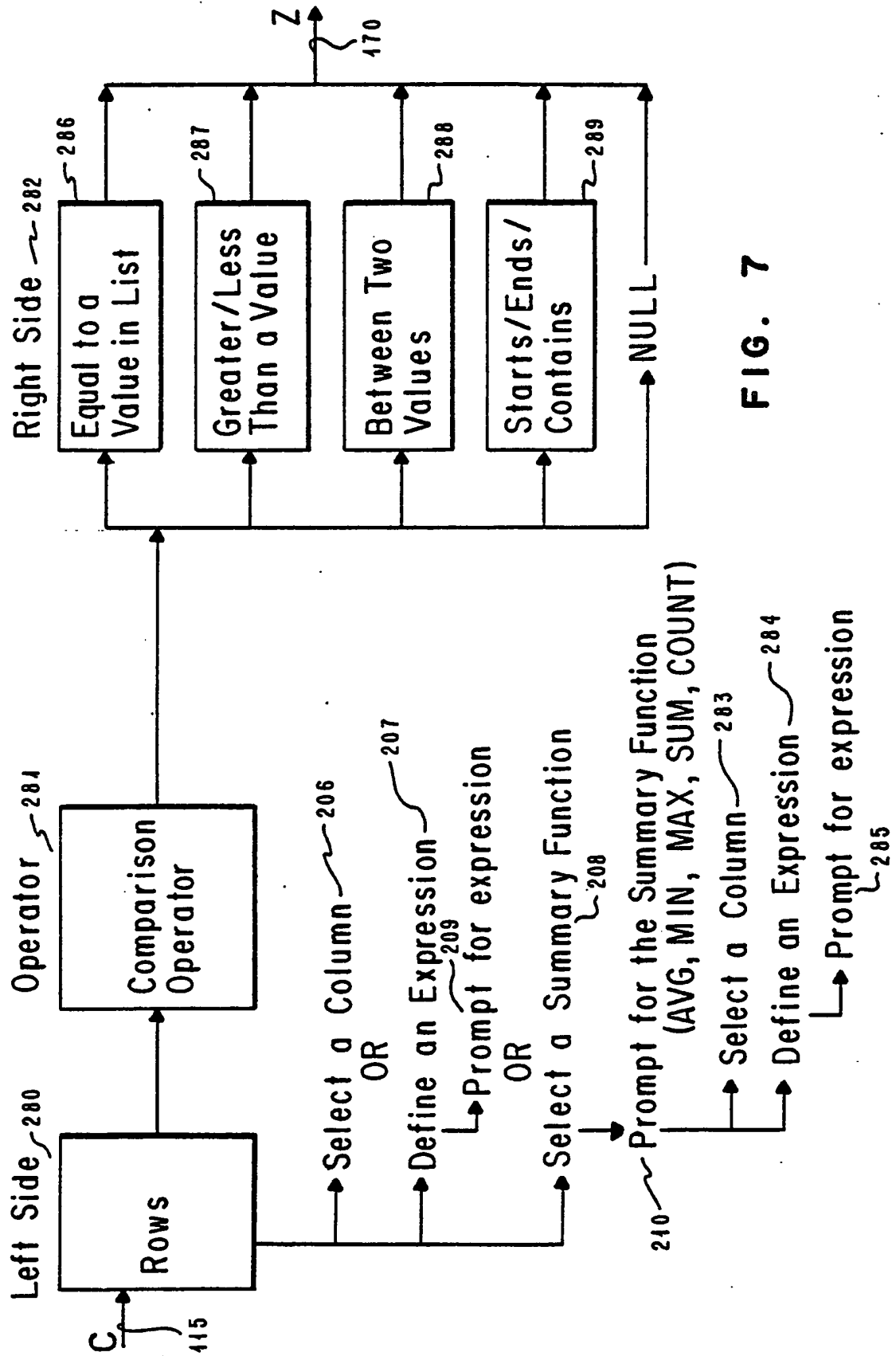


FIG. 7

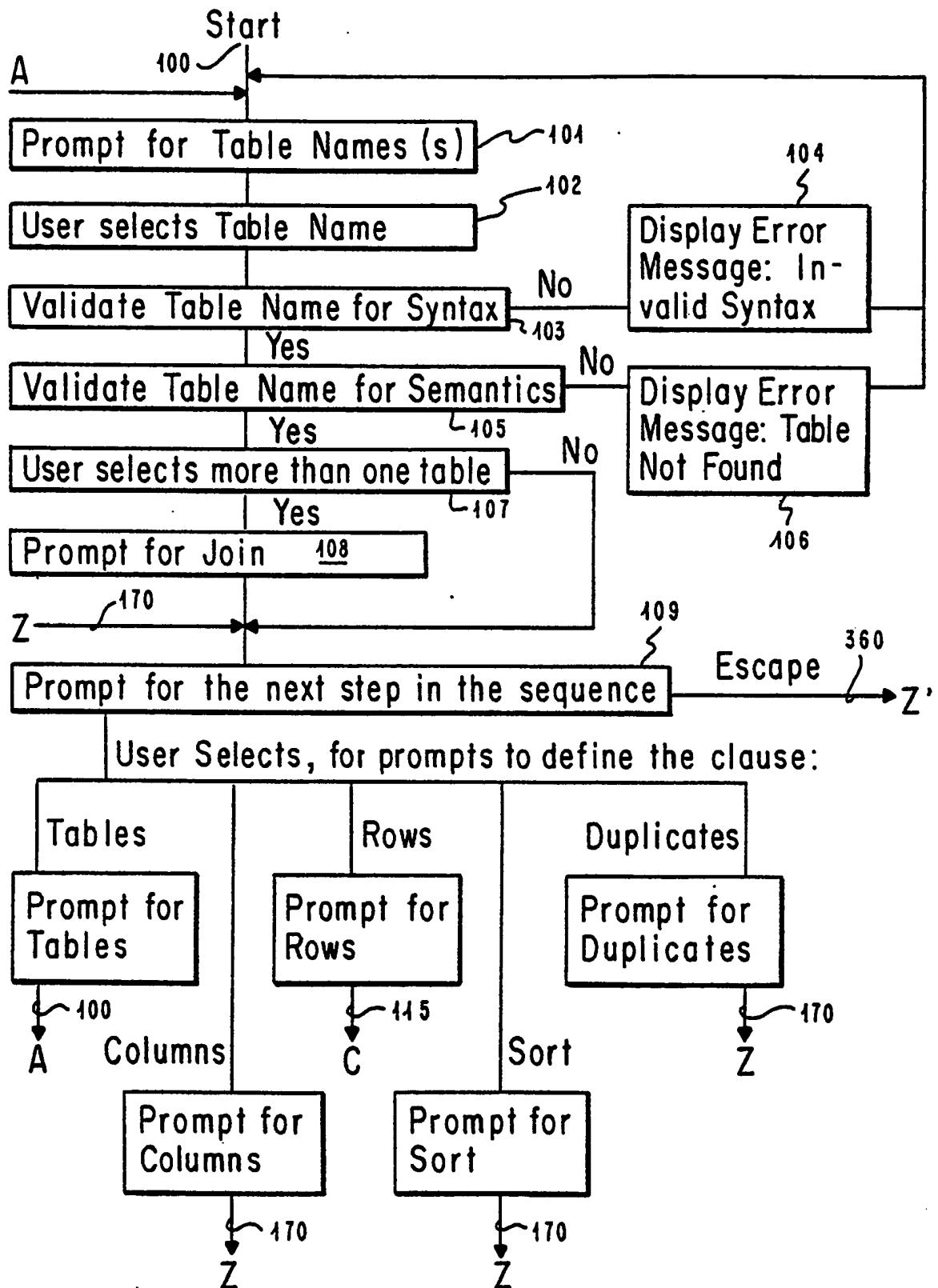


FIG. 8A

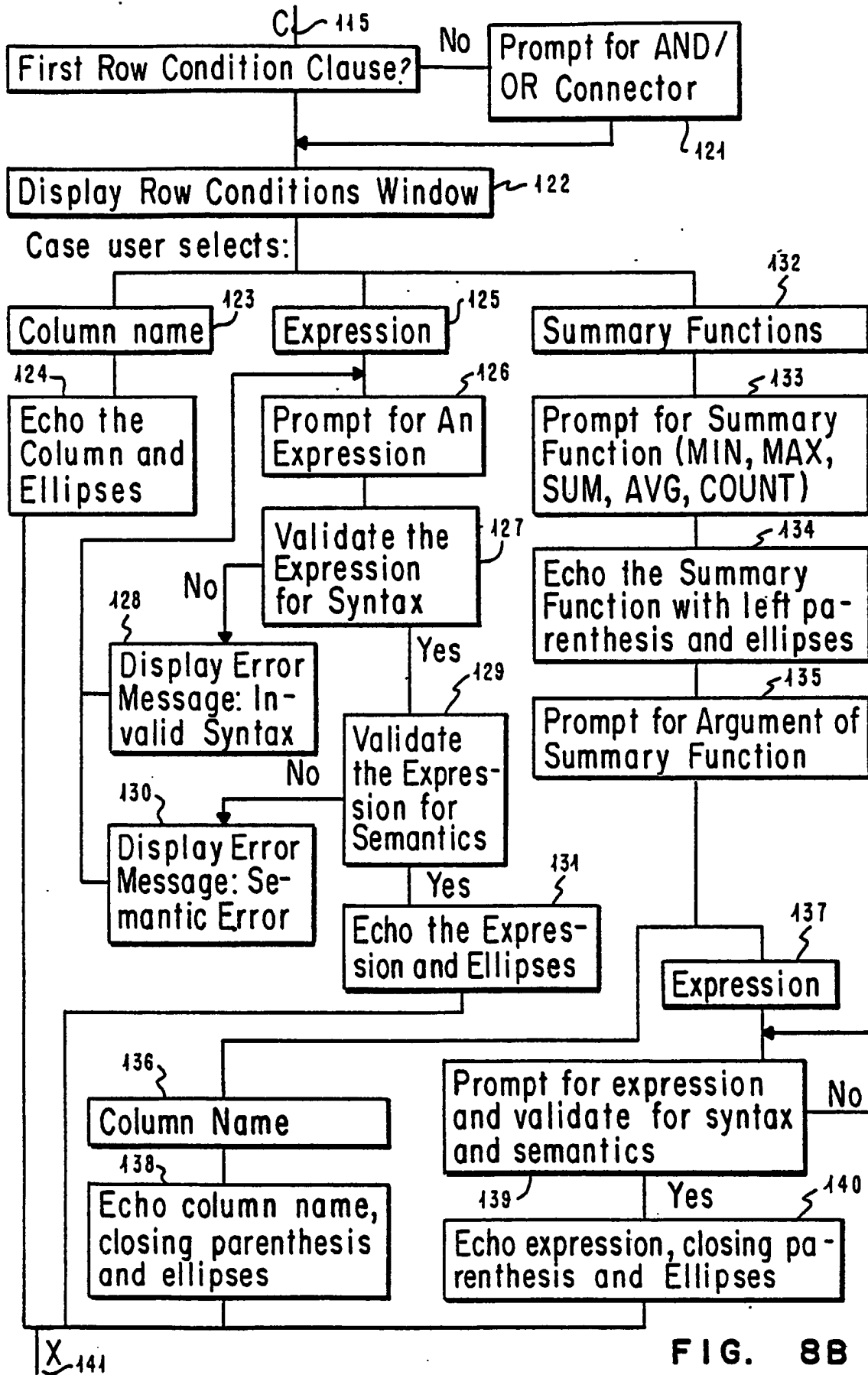


FIG. 8B

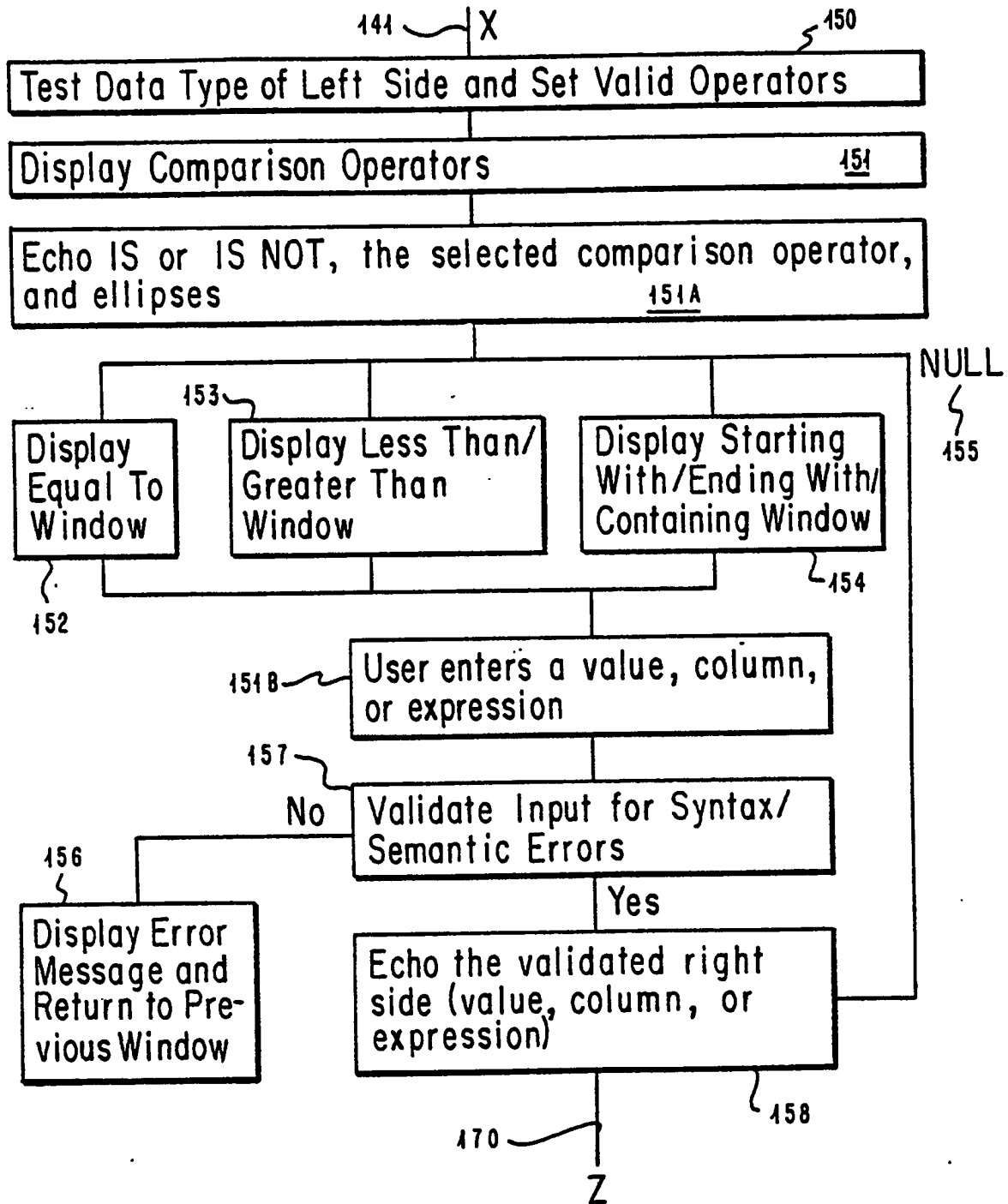


FIG. 8C

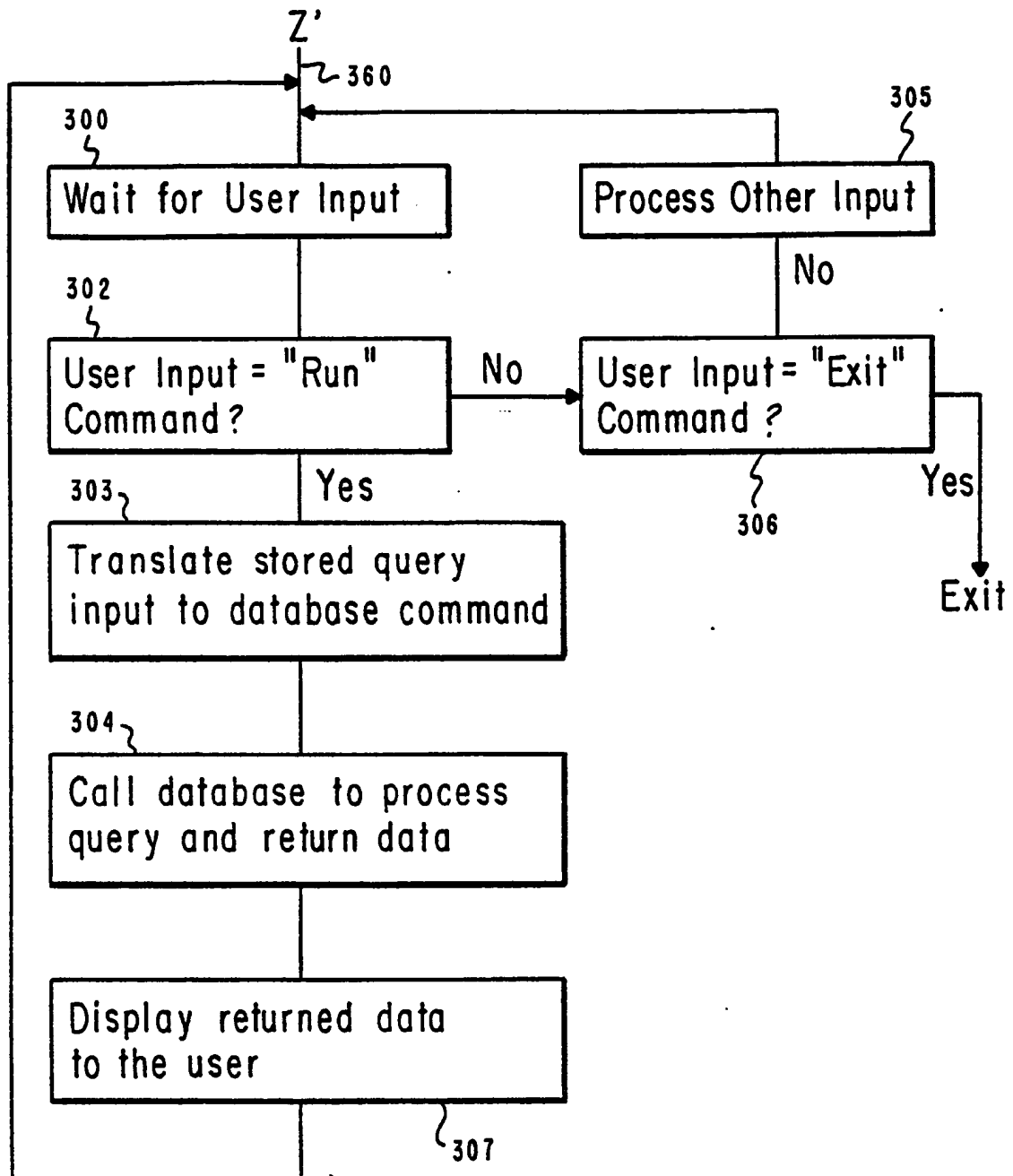


FIG. 8D

⁹⁷ SQL <u>KEYWORDS</u>	⁹⁸ USER-FRIENDLY NON- <u>SQL KEYWORD</u>
SELECT — — — →	COLUMNS
DISTINCT — →	DUPLICATES
FROM — — — →	TABLES
WHERE — — — →	ROW CONDITIONS
GROUP BY — — →	(implied when summary functions used)
HAVING — — — →	ROW CONDITIONS
ORDER BY — — →	SORT

FIG. 9

²⁰ ¹⁸ ²¹ ¹⁶ ¹⁷ ²² ²³
 SELECT [ALL | DISTINCT] { * | select-list }
 where a select-list is defined as:
⁴⁰
 1. Constant ⁴⁰
 2. Column Name ⁴¹
 3. Arithmetic Expression ⁴²
 4. Column Function where column functions is defined as:
⁵⁰ ⁴³
⁵¹ ~ AVG (expression) ⁴⁴
⁵² ~ COUNT (*) | COUNT (DISTINCT colname) ⁴⁵ ⁴⁶
⁵³ ~ MIN (expression) ⁴⁷
⁵⁴ ~ MAX (expression) ⁴⁸
¹¹ ⁵⁵ ²⁴ ~ SUM (expression) ⁴⁹ ¹⁹
 FROM tablename [correlname] [,tablename [correlname]]..
 [WHERE predicate] ¹² ²⁶ ²⁵
 where the syntax for a predicate is defined as:
 expression1 ⁶² operator ⁶¹ expression2 ⁶⁰
 where operators are defined as:
⁶³ ~ = equal to
⁶⁴ ~ < > not equal to
⁶⁵ ~ > greater than
⁶⁶ ~ < less than
⁶⁷ ~ > = greater than or equal to
⁶⁸ ~ < = less than or equal to
⁶⁹ ~ BETWEEN between two values
 where the syntax for BETWEEN is defined as:
⁷⁰ ⁷¹ ⁷²
 expression1 [NOT] BETWEEN expression2
 AND expression3 ⁷³
⁷⁴ ~ IN in a list of values
 where the syntax for IN is defined as:

FIG. 10A

⁷⁵expression1 ⁷⁶[NOT] ⁷⁷IN ⁷⁸expression2
⁷⁸expression1 ⁷⁹[NOT] ⁸⁰IN (value1 [,value2]...)

⁸⁴IS NULL equal to a null value
 where the syntax for NULL is defined as:
⁸²colname ⁸³IS ⁸³[NOT] NULL

⁸⁴LIKE matches a pattern
 where the syntax for LIKE is defined as:
⁸⁵colname ⁸⁶[NOT] ⁸⁷LIKE ⁸⁷expression

¹³[GROUP BY ²⁷colname [, colname]...]

¹⁴[HAVING ²⁸predicate]

¹⁵[ORDER BY ²⁹{colname | colnum ³⁰[ASC | DESC]}
³¹{colname | colnum [ASC |
 DESC]} ...]]

FIG. 40B

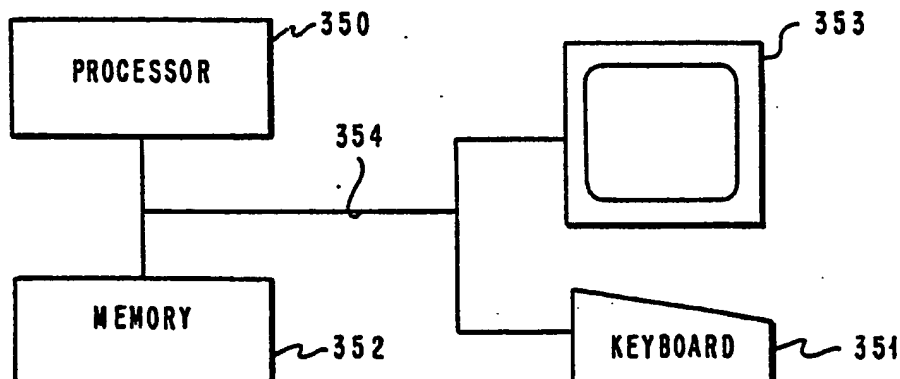


FIG. 41

(19)



Europäisches Patentamt
European Patent Office
Office européen des brevets



(11) Publication number:

0 336 584 A3

(12)

EUROPEAN PATENT APPLICATION(21) Application number: **89302582.5**(51) Int. Cl.⁵: **G06F 15/40**(22) Date of filing: **16.03.89**(30) Priority: **07.04.88 US 179181**(43) Date of publication of application:
11.10.89 Bulletin 89/41(84) Designated Contracting States:
CH DE ES FR GB IT LI NL SE(88) Date of deferred publication of the search report:
30.09.92 Bulletin 92/40(71) Applicant: **International Business Machines Corporation**
Old Orchard Road
Armonk, N.Y. 10504(US)(72) Inventor: **Chang, Philip Yen-Tan**
6221 Ledge Mountain Drive
Austin Texas 78731(US)
Inventor: **Coyle, Daniel Jerome, Jr.**
2003 Ploverville Lane
Austin Texas 78728(US)
Inventor: **Malkemus, Timothy Ray**
1602 Rock Creek Drive
Round Rock Texas 78681(US)
Inventor: **Rodriguez, Rebecca Ann**
606 Willowwood Lane
Pflugerville Texas 78660(US)
Inventor: **Welti, Philip John**
1305 Oakridge Drive
Round Rock Texas 78681(US)(74) Representative: **Balley, Geoffrey Alan**
IBM United Kingdom Limited Intellectual
Property Department Hursley Park
Winchester Hampshire SO21 2JN(GB)(54) **Sort merge output.**

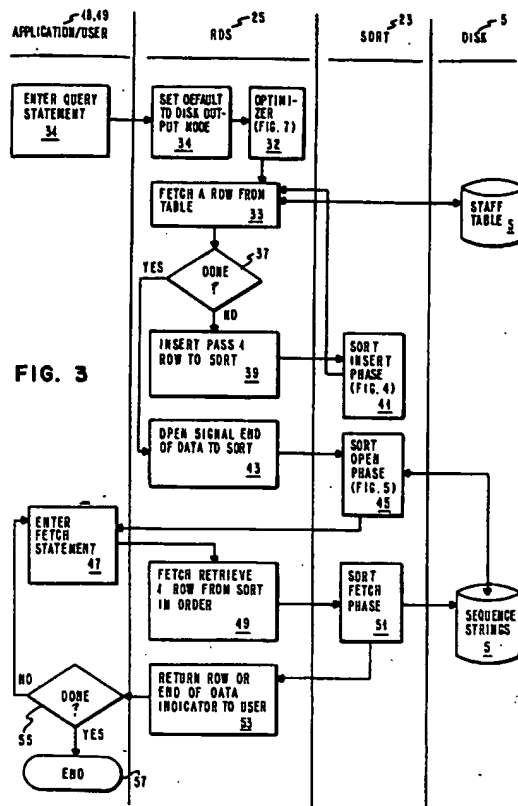
(57) In a relational database system, a method is used that increases the performance of the sort operation. An optimiser routine in the relational database manager analyses a user's complete query to determine whether the final sorted results can be used directly by the user as sorting occurs. If the sort results can be used, the sort results are sent to Relational Data Services in the relational database manager for output to the user.

Depending upon the determination made by the optimiser routine, one of two output modes for the final sorted sequence string of data are selected by the relational database manager. In disk output

mode, the last pass of the final sorted sequence string is written to disk. In fast direct output mode, the records of data are sent to the user as the final sorted sequence string is being merged during the last pass.

Such an arrangement reduces the total sort time by eliminating the overhead of writing to disk during the fast direct output mode. Also, the response time, or availability for each record is reduced since each record is retrieved or sent to the Relational Data Services immediately after it has been sorted into the final sort order instead of completing the entire sort first, and then writing to disk.

EP 0 336 584 A3





European Patent
Office

EUROPEAN SEARCH REPORT

Application Number

EP 89 30 2582

DOCUMENTS CONSIDERED TO BE RELEVANT			
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl.4)
X	THE 12TH ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE 17 June 1985, BOSTON, USA pages 250 - 257; S. KAMIYA ET AL: 'A Hardware Pipeline Algorithm for Relational Database Operations and Its Implementation Using Dedicated Hardware' * page 251, column 2, line 1 - page 251, column 2, line 46; figure 2 * * page 254, column 2, line 36 - page 256, column 2, line 10 * ---	1,9-10	G06F15/40
X	THE 11TH INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE 5 June 1984, ANN ARBOR, USA pages 134 - 141; D. GAJSKI ET AL: 'A PARALLEL PIPELINED RELATIONAL QUERY PROCESSOR: AN ARCHITECTURAL OVERVIEW' * summary * * page 135, column 2, line 24 - page 138, column 2, line 44 *	1	
A	---	2-10	TECHNICAL FIELDS SEARCHED (Int. Cl.4)
A	EP-A-0 066 061 (TOKYO SHIBAURA DENKI K.K.) * page 10, line 24 - page 17, line 7 * -----	1-10	G06F
The present search report has been drawn up for all claims			
Place of search THE HAGUE		Date of completion of the search 03 AUGUST 1992	Examiner FOURNIER C, D, J.
CATEGORY OF CITED DOCUMENTS X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technological background O : non-written disclosure P : intermediate document T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date D : document cited in the application L : document cited for other reasons ***** & : member of the same patent family, corresponding document			



(12) **EUROPEAN PATENT APPLICATION**

(43) Date of publication:
20.11.1996 Bulletin 1996/47

(51) Int Cl.⁶: **G06F 17/30**

(21) Application number: **96303413.7**

(22) Date of filing: **14.05.1996**

(84) Designated Contracting States:
DE FR GB

(30) Priority: **19.05.1995 US 446170**

(71) Applicant: **AT&T IPM Corp.**
Coral Gables, Florida 33134 (US)

(72) Inventors:
 • **Jagadish, Hosagrahar Visvesvaraya**
Union, New Jersey 07922 (US)

• **Silberschatz, Abraham**
Summit, New Jersey 07901 (US)
 • **Mumick, Inderpal Singh**
Union, New Jersey 07922 (US)

(74) Representative:
Buckley, Christopher Simon Thirsk et al
Lucent Technologies,
5 Mornington Road
Woodford Green, Essex IG8 0TU (GB)

(54) **Method for querying incrementally maintained transactional databases**

(57) The chronicle data model of the present invention permits the capture, within the data model, of many computations common to transactional recording systems. An important aspect of the chronicle data model is incremental maintenance of materialized view in time independent of the size of the recorded stream.

The chronicle data model differs from the relational model in that it encapsulates within itself (1) support for aggregate, or summary, queries over sequences that may not be stored in their entirety, through the use of materialized (or persistent) views; and, (2) automatic incremental maintenance of persistent views as records

are inserted into the sequence.

A chronicle data model can be implemented on top of a relational system. An aspect of this invention is to reduce the complexity of the application domains that need a chronicle system by encapsulating this model within the database. Within the chronicle data model of the present invention is disclosed: the type of summary queries that can be answered by using persistent views is determined, the complexity of incrementally maintaining the persistent views and development of a language ensuring low maintenance complexity independent of the sequence sizes.

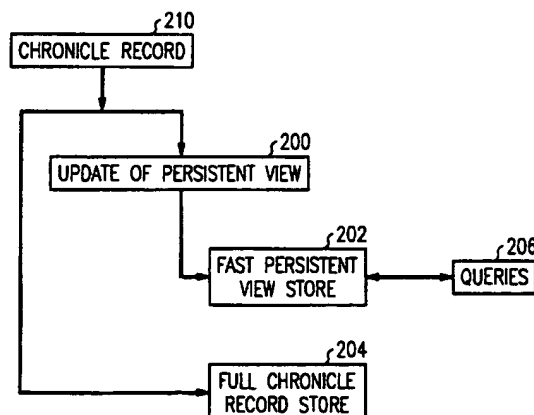


FIG. 2

Description

TECHNICAL FIELD

This invention relates generally to database systems, and more particularly, to transactional recording systems.

BACKGROUND OF THE INVENTION

Many database systems are used to record a stream of transactional information, such a credit card transactions, telephone calls, stock trades, flights taken, sensor outputs in a control system, etc. Applications such as banking, billing, and trading that deal primarily with transactional data have several common characteristics.

Each record in an incoming sequence of transaction records has several attributes of the transaction. The sequence of records can be very large, and grows in an unbounded fashion. The transaction records are stored in a database for some latest time window, but it is beyond the capacity of conventional database systems to store and provide access to this sequence for ever.

For example, a major telecommunications company is known to collect 75GB (giga-bytes) of sequence data every day, or 27TB (tera-bytes) of sequence data every year. No current database system can even store so much data, far less make it accessible in an interactive manner.

Queries are conducted over the stored sequence of transaction records, with stringent response time requirements. Of particular interest are summary queries that access a summarization, or aggregate information of past transactional activity.

For example, a cellular phone company may want to make a summary query that computes the total number of minutes of calls made in the current month from a particular cellular phone number. It may be desirable to make such a query every time a cellular phone is turned on, and to display the result on the customer's phone instrument. A customer care agent in the cellular company may want to ask a different summary query, such as: "What is the total number of minutes of calls made from a given number since the number was assigned to the current customer?"

These applications can be and are implemented using commercially available relational databases. However, the relational model is not suitable to capture and exploit the peculiar characteristics of a transaction recording system. For example, there is no support for answering a summary query over a sequence that is not stored in the database in its entirety. There is no support for answering a summary query over a large sequence, even if the sequence is stored, with the speed needed to process a banking transaction, or to display the answer on the customer's phone at power-on time.

These summary queries are therefore supported in

today's systems by procedural application code. For example, an application program may define a few summary fields (e.g., *minutes_called*, *dollar_balance*) for each customer, and update these fields whenever a new transaction is processed for this purpose. Summary queries are then answered by looking up the summary fields, rather than going into the sequence of transaction records. This gives the applications a fast response time, as well as independence from the need to lookup past transactional data. Some applications, such as ATM withdrawals, require that a summary field (*dollar_balance*) be updated as the transaction is executed, since the summary query needs to be made before the next ATM withdrawal. Some applications may choose to use triggers to invoke the updating code; others may update the summary fields as they process the transaction in batch. In all cases the logic to update the summary fields due to a transaction is encoded procedurally, and the burden of writing this code lies with the application programmer. This updating code is known to be very tricky, and has been the cause of well-publicized banking disasters (e.g. Chemical bank ATM withdrawals caused incorrect updates on February 18, 1994, leading to thousands of bounced checks and frustrated customers).

The need to define summary fields and to write the update procedures within the application code is one of the reasons for the complexity of banking, billing, and other similar systems.

SUMMARY OF THE INVENTION

The present invention discloses a method for using a computer system to process queries of a transactional recording system database by first defining a chronicle comprising a relation with an additional sequencing attribute, where the relation comprises an unordered set of tuples; then generating a persistent view from the chronicle using a view definition language. The chronicle is incrementally maintained in response to transactions processed in the database. Summary queries are answered from the persistent view.

The persistent view of the present invention is generated by first generating the chronicle by an intermediate chronicle algebra and then summarizing and mapping the chronicle produced by the intermediate chronicle algebra into the persistent view.

The persistent views of the present invention are incrementally maintained by proactively updating a relation external to the chronicle by insertion, deletion or modification of a tuple in the unordered set of tuples of the relation. The relation has a plurality of temporal versions, each one for every update. The persistent view on the chronicle may be defined by joining the tuples of the chronicle with one of the plural temporal versions of the relation. The chronicle is updated by inserting tuples into the chronicle, then the persistent view which depends on this chronicle is updated to reflect this inser-

tion. The updated persistent view is then stored.

One aspect of the present invention provides for an intermediate chronicle algebra used to derive the persistent view, which includes operations for selecting from the tuples on the chronicle those which satisfy a predicate; projecting the chronicle on attributes that include a sequencing attribute; forming a natural equijoin between two chronicles on the sequencing attribute; forming a union of two chronicles in a same chronicle group, and having a same type; determining a difference of two chronicles in the same chronicle group, and having the same type; deriving a groupby with aggregation, having grouping attributes, with a sequence number as one of the grouping attributes; and, determining a cartesian product between the chronicle and any relation, wherein an implicit temporal join on the sequencing attribute exists between the chronicle and the relation.

Another aspect of the invention allows for generating a plurality of persistent views for the chronicle and incrementally maintaining the plurality of persistent views.

Yet another aspect of the invention allows for the periodic maintenance of a persistent view whether the persistent view is maintained for a specified period and is overwritten at end of that period.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1A depicts operations performed at a time of new chronicle record entry according to one embodiment of the present invention.

Fig. 1B depicts operations performed at query time according to one embodiment of the present invention.

Fig. 2 illustrates periodic persistent view updating according to one embodiment of the present invention.

Fig. 3 shows an illustrative embodiment of an implementation of the present invention.

DETAILED DESCRIPTION

Some known preliminary matters are now set out which are helpful to an understanding of the present invention.

Under relational algebra, and with grouping and aggregation operations of Structured Query Language ("SQL"), the following syntax is used to express a grouping operation:

GROUPBY(*R*,*GL*,*AL*)

where *R* is the relation being grouped, *GL* = [*G*₁,...,*G*_{*m*}] is the list of grouping attributes, and *AL* = [*A*₁,...,*A*_{*n*}] is the list of aggregation functions. This expression defines a result relation with attributes in *GL* and with one attribute for each aggregation function. Only those aggregation functions *A_j* that are incrementally computable, or are decomposable into incremental computation

functions are pertinent to discussion of the present invention. For the complexity analysis, it is assumed that each aggregation function can be computed in time *O* (*n*) over a group of size *n*, and can be incrementally computed in time *O*(1) over an increment of size 1. **MIN**, **MAX**, **SUM** and **COUNT** are examples of such functions.

The present invention will now be described with respect to specific embodiments thereof.

Figure 1A illustrates one embodiment of the operations performed at a time of new chronicle record entry. An existing persistent view is derived in step 100 by the process described below. A new chronicle record is entered in step 101. The existing persistent view 100 is incrementally maintained in step 102 by the procedure as described below, generating a new persistent view 104.

In Fig. 1B one embodiment of the operations performed at a time of query are shown. A query is posed against the chronicle in step 106. The query is answered either exclusively from or with the aid of the persistent view in step 108.

A chronicle database according to the present invention consists of *relations*, *chronicles*, and *persistent views*. Relations are standard, and each relation may have several temporal versions.

A *chronicle* is similar to a *relation*, except that a chronicle is a *sequence*, rather than an unordered set, of tuples. A chronicle can be represented by a relation with a *extra sequencing* attribute, whose values are drawn from an infinite ordered domain. An existing attribute of the relation can be marked as the sequencing attribute or an additional sequencing attribute can be added. The only update permissible to a chronicle is an insertion of tuples, with the sequence number of the inserted tuples being greater than any existing sequence number in the chronicle. There is no requirement that the sequence numbers be dense. Chronicles can be very large, and the entire chronicle may or may not be stored in the system.

There is a temporal instant (or "chronon") associated with each sequence number. All operations on a tuple of a chronicle are with respect to the database as it was at that point in time. Thus, any join of a chronicle *C* and a relation *R* is a union of the corresponding joins of each tuple in *C* with the version of *R* that existed at the temporal instant of the tuple in *C*.

The database maintains a fixed number of *persistent views*, which are views that are materialized into relations, and are always maintained current in response to changes to the underlying database. Each persistent view is materialized when it is initially defined, and it is kept up-to-date, reflecting all the changes that occur in the database as soon as these changes occur. Of particular concern is the maintenance of a persistent view after every append to a chronicle.

Persistent views are defined in a view definition language *L*, and correspond to the procedurally computed

summary fields in the current transaction system. Each choice of L derives a particular instance of the chronicle data model of the present invention.

A chronicle database system is a quadruple:

$$(C, R, L, V)$$

where $C = \{C_0, C_1, \dots, C_n\}$, $n \geq 0$, is a set of chronicles, $R = \{R_0, R_1, \dots, R_m\}$, $m \geq 0$, is a set of relations, and $V = \{V_0, V_1, \dots, V_1\}$, $1 \geq 0$, is a set of persistent views defined in a language L .

Queries that access the relations and persistent views can be written in any language - relational algebra, structured query language ("SQL"), Datalog, etc. The choice of this language is orthogonal to the chronicle model. The chronicle model enables fast response to queries that access the persistent views; these queries may otherwise have been defined as complex SQL queries over the relations and chronicles and thus would not likely have been answerable with acceptable performance. Further, a system would typically provide detailed queries over some latest window on the chronicle; again the choice of the window and the query language are orthogonal to the chronicle model.

The only types of updates permitted are those that modify the relations and chronicles. An update to a relation R can be an insert, delete, or modification of a tuple in R . An update to a chronicle C can only be the insertion of a new tuple (or tuples) to C with a sequence number greater than the sequence number of all existing tuples in C . These updates are discussed individually below.

Each relation conceptually has multiple temporal versions, one after every update. In any persistent view defined in language L , any joins between the relations and chronicles have an implicit temporal join on the sequencing attribute: each tuple of a chronicle is joined with the version of the relations that was current when the particular chronicle tuple was generated.

If an update to a relation affects only the versions corresponding to sequence numbers not seen as yet, then it is a *proactive* update. Such an update does not affect the persistent views. Only subsequent chronicle updates see the new relation values. Since maintainability of persistent views is critical in the chronicle model, the language L is limited so that only proactive updates to relations are allowed.

In contrast, a *retroactive* update to a relation would require older tuples in the chronicles to be re-processed. Such updates, when necessary, are computationally expensive to maintain, may not be maintainable if the entire past chronicle is unavailable, and are not included as part of the chronicle model.

An update to a chronicle may cause a change in each persistent view, the maintenance of which is discussed below.

Each time a transaction completes, a record for the

transaction is appended to the chronicle, which may affect the state of one or more persistent views. The transaction rate that can be supported by a chronicle system is determined by the complexity of incremental maintenance of its persistent views. Thus, it is important to choose a language L that ensures that incremental view maintenance can be done efficiently. Moreover, since chronicles may not be stored in the system, the language L should allow the incremental maintenance without having access to the entire chronicles. Ideally, the complexity of maintaining a view defined in L should be low - independent of the size of the relations and the view itself, modulo the overhead of index lookups.

In the present invention, the complexity of a chronicle system is the complexity of incremental computation of the language L used to express the persistent views. A class IM- T means that all persistent views defined in the language can be maintained in time $O(T)$ in response to a single append into a chronicle (IM for incremental maintenance).

The following incremental complexity classes are defined in one embodiment of the present invention:

- IM-Constant: A language is in the class IM-Constant if any persistent view defined in the language can be incrementally maintained in response to a single append into the chronicle in constant time.
- IM-log(R): A language is in the class IM-log(R) if any persistent view defined in the language can be incrementally maintained in response to a single append into the chronicle in time logarithmic in size of the relations.
- IM- R^k : A language is in the class IM- R^k if any persistent view defined in the language can be incrementally maintained in response to a single append into the chronicle in time polynomial in size of the relations.
- IM- C^k : A language is in the class IM- C^k if any persistent view defined in the language can be incrementally maintained in response to a single append into the chronicle in time polynomial in size of the chronicle.

The size of the relations, $|R|$, is assumed to be much smaller than the size of the chronicle $|C|$. It will be easily understood that the following relationships hold between the sets of views that can be described by languages in each class:

$$\text{IM-Constant} \subset \text{IM-log}(R) \subset \text{IM-}R^k \subset \text{IM-}C^k$$

In a high throughput system, a complexity of IM-Constant is desired which implies that even index

lookups are not permitted, and is thus difficult to achieve. At the other end of the spectrum, a chronicle model with complexity IM- C^* would permit arbitrary access to the chronicle. Such complexity is totally impractical for an operation to be executed at each append into each chronicle.

The choice of language L for defining persistent views with a low IM complexity is crucial. Relational algebra with grouping and incrementally computable aggregate operators is not an acceptable choice for L , because relational algebra, extended with grouping and aggregation, applied to chronicles and relations, is in the class IM- C^k , and is not in the class IM- R^k .

The largest restrictions of relation algebra are in the first three IM complexity classes, which are all independent of the size of the chronicle are derived. The derivation can be broken down into two steps. First, defining an intermediate chronicle algebra that maps chronicles and relations into chronicles, Second, adding a summarization step that maps chronicle algebra expressions into relations by projecting out the sequencing attribute, possibly doing a grouping and aggregation alongside.

All inserted tuples into a chronicle must have a sequence number greater than all existing sequence numbers, but multiple tuples with the same sequence number can be inserted simultaneously. For instance, when a tuple is inserted into a base chronicle, each of the two operands of a union may derive a distinct tuple with the same sequence number. The union expression can then have two distinct tuples with the same sequence number.

A *chronicle group* is defined in the present invention as a collection of chronicles whose sequence numbers are drawn from the same domain, along with the requirement that an insert into any chronicle in a chronicle group must have a sequence number greater than the sequence number of any tuple in the chronicle group. Operations like union, difference and join are permitted amongst chronicles of the same chronicle group.

One illustrative embodiment of chronicle algebra ("CA") according to the present invention consists of the following operators (C is a chronicle or a chronicle algebra expression, A_1, \dots, A_n are attributes of the chronicle, and p is a predicate):

- A selection of a chronicle, $\sigma_p(C)$, where p is a predicate of the form $A_1 \theta A_2$, or $A_1 \theta k$, or a disjunction of such terms, k is a constant, and θ is one of $\{=, \neq, \leq, <, >, \geq\}$. $\sigma_p(C)$ selects chronicle tuples that satisfy the predicate p . The resulting chronicle has the same type as the chronicle C .
- Projection of a chronicle on attributes that include the sequencing attribute, $\Pi_{A_1, \dots, A_n}(C)$.
- A natural equijoin between two chronicles on the sequencing attribute, $C_1 \bowtie_{C_1.SN=C_2.SN} C_2$ where SN is the sequencing attribute, C_1 and C_2 are chronicles in the same chronicle group, and one of the sequencing attributes is projected out from the re-

sult.

- Union of two chronicles, $C_1 \cup C_2$, where C_1 and C_2 are chronicles in the same chronicle group, and have the same type.
- Difference of two chronicles, $C_1 - C_2$, where C_1 and C_2 are chronicles in the same chronicle group, and have the same type.
- A groupby with aggregation, with the sequence number as one of the grouping attributes:
GROUPBY(C, GL, AL), where C is a chronicle being grouped, GL is the list of grouping attributes (must include the sequencing attribute), and AL is the list of aggregation functions.
- A cartesian product between a chronicle, C , and any relation R , $C \times R$. Though this operation is written as a cross product, an implicit temporal join on the sequencing attribute exists between C and R .

A view defined by the chronicle algebra is monotonic with respect to insertions into the base chronicles. Whenever tuples with sequence numbers greater than all existing sequence numbers are added to the base chronicles, the effect is to add tuples with some of these new sequence numbers to the view.

Each view defined using a chronicle algebra expression is a chronicle in the same chronicle group as the operand chronicles. CA(1) is chronicle algebra, without the cross product operation between chronicles and relations. CA(\bowtie) is chronicle algebra, where the cross product operation between chronicles and relations is replaced by a join, with a guarantee that at most a constant number of tuples join with each chronicle tuple. A sufficient condition for the guarantee is that the join be on a key of the relation R .

The changes, due to insertions into the base chronicles, for a chronicle view defined by CA can be computed in time and space independent of the size of the chronicles and independent of the size of the view with Time = $O((u | R |) \log(|R|))$, and Space = $O((u | R |))$, where u is the number of unions in the expression defining the view, $|R|$ is the size of the relation R , and j is the number of equijoins and cross products in the expression defining the view. Those defined by CA(\bowtie) can be computed in Time = $O(u | \log(|R|))$, and Space = $O(u |)$. Those defined by CA(1) can be computed in Time = $O(u |)$, and Space = $O(u |)$.

In all cases, neither the chronicle view nor the chronicles need to be stored or accessed for the view maintenance; and this is the reason for obtaining a complexity which is independent of both the sizes of the chronicle and the sizes of the view.

Any extension of the chronicle algebra with either of (1) projection without including the sequencing attribute, or (2) a groupby operation without including the sequencing attribute as a grouping attribute leads to an algebra that can define an expression that is not a chronicle. Further, an extension of the chronicle algebra with either of (1) cross product between chronicles, or (2) a

non-equi-join between two chronicles leads to an algebra that can define an expression for which the time for incremental view maintenance is dependent on the size of the chronicle.

It is implied by this that the chronicle algebra is the largest subset of relational algebra operations that is in $IM-R^k$, and the $CA(\bowtie)$ is the largest subset of relational algebra operations that is in $IM-log(R)$. Expressions can also be defined using the cross-product or non-equi-join between chronicles that are in $IM-R^k$.

A summarization step maps chronicles produced by chronicle algebra into persistent views, which are relations without the sequencing attributes. The persistent view is then stored, and updated whenever an insert occurs into the chronicles on which the persistent view depends.

The *summarized chronicle algebra* (SCA), has the two basic operations that can eliminate the sequence attribute and map a chronicle algebra expression χ into a relation.

- Projection, with the sequencing attribute projected out; that is, $\Pi_{A_1, \dots, A_n}(X)$, where the attributes A_1, \dots, A_n do not include the sequencing attribute.
- Grouping with aggregation, where the sequencing attribute is not included in the grouping list, and where the aggregation functions are incrementally computable (or decomposable into incrementally computable functions); represented as $GROUP-BY(\chi, GL, AL)$ where the grouping list GL does not include the sequencing attribute of χ .

If the expression χ is in the $CA(1)$, the resulting language is called $SCA(1)$; if the expression χ is in the $CA(\bowtie)$, the resulting language is called $SCA(\bowtie)$.

It follows that every persistent view expressed in SCA produces a relation (not a chronicle) that does not have the sequence number as an attribute. Once a relation is defined using SCA, it could be further manipulated by using relational algebra and the other relations in the system, to define a persistent view. However, since incremental maintenance is the key, it is important to store a persistent view that can be maintained without accessing the full chronicles over which the summarization step is defined.

Given a set of changes to chronicle algebra expression, incremental maintenance of a persistent view written in SCA in response to insertions to a chronicle can be done in

- Space equal to the size of the view.
- Time = $O(t \log(|V|))$, where $|V|$ is the size of the persistent view V , and t is the number of tuples inserted into the chronicle algebra (or $CA(1)$ or $CA(\bowtie)$) expression χ .

SCA is contained in class $IM-R^k$. $SCA(\bowtie)$ is contained in class $IM-log(R)$, and $SCA(1)$ is contained in

class $IM-Constant$.

Thus, although the result of a chronicle algebra expression contains sequence numbers, and therefore may have a size that is polynomial in $|C|$, incremental maintenance of summarized chronicle algebra expressions can be done in time independent of $|C|$, since the chronicle view is not accessed during maintenance.

Some applications within the purview of the chronicle model require the definition of a view that is computed over several, potentially overlapping, intervals on a chronicle, one view computation for each interval. To address this need, a *periodic summarized chronicle algebra* is implemented by adding, to the chronicle algebra, features to construct sets of time intervals over which the persistent views can be computed. A mapping from sequence numbers in a chronicle to time intervals must be made for the periodic summarized chronicle algebra to be defined.

Given a view V in summary algebra, and a calendar D (i.e., a set of time intervals), $V < D >$ specifies a set of views V_1, \dots, V_n , one for each interval in the calendar D . The view V_i for interval i is defined as in V , but with a selection on the chronicle, which requires that all chronicle tuples be within the interval i , under the mapping defined from sequence numbers to time intervals. If the calendar D has an infinite number of intervals, there will be an infinite number of views V_i . The view expression $V < D >$ is called a periodic view. When the calendar D has only one interval, the periodic view corresponds to a single view defined using an extra selection on the chronicle.

The periodic summarized chronicle algebra also provides for an expiration time for a view, after which the view is not needed. Expiration dates allow the system to implement an infinite number of periodic views, provided only a finite number of them are current at any one instant.

Figure 2 illustrates periodic updates according to one embodiment of the present invention. An updated persistent view is generated from a chronicle record 210 in step 200. The updated persistent view is stored in an efficient ("fast") storage structure in step 202. Queries are answered against the fast persistent view in step 206. The chronicle record is stored in a full chronicle store in step 204, appended to previously stored chronicle records.

Billing applications typically require periodic views over nonoverlapping intervals. The evaluation of these can be optimized by starting to maintain a view as soon as its time interval starts, and stopping its maintenance as soon as its interval ends. Periodic views over overlapping intervals can be defined to compute moving averages over the transactions in a chronicle. For example, consider a periodic view for every day that computes the total number of shares of a stock sold during the 30 days preceding the day. The computation of these views can be optimized by noticing that the sum of shares is an incrementally computable function. The

total number of shares sold for each of the last 30 days are kept separately, and the view is derived as the sum of these 30 numbers. Moving from one periodic view to the next one involves shifting a cyclic buffer of these 30 numbers. Further, if an expiration date is given, the space for the periodic view can be rescued.

A computation is automatically derived by the system for a generic periodic view expressed over any given set of overlapping time intervals.

When multiple views are to be maintained over the same chronicle, each update to the chronicle requires checking all the views to determine if they need to be updated in the following manner:

- Identify the persistent views *V* that will be affected. These are filtered out early so as not to waste computation resources. The problem is similar to detecting the active rules that must be checked after a database update.
- For each persistent view *V*, identify the tuples that will be affected. Thus, the persistent views need to have indices.
- When periodic views are used, identify the persistent views that are *active* - these are the views defined for the *current* time interval, and only these periodic views need to be maintained upon insertions into the chronicle.

Efficient storage structures are needed for fast access to the updated persistent view tuples. Normally only the updated persistent view will be stored. "Slow cheap storage devices" can be used where it is desired to store superseded persistent views (i.e., access speed is not critical so a conventional tape drive or the like will suffice).

Applications often define computations applying to a batch of transactions. For example, a bank may charge a fee based upon the total number of transactions within a period, a telephone company offers a discount based upon the total calls within a period, an airline gives bonus miles based upon total activity within a certain period. For example, a popular telephone discounting plan in the USA gives a discount of 10% on all calls made if the monthly undiscounted expenses exceed \$10, a discount of 20% if the expenses exceed \$25, and so on. In such applications, a common assumption is that the computations are performed once at the end of the period. This leads to two problems: first, the results of these computations are either available after or inaccurate before the end of the period over which the discount applies; and, second, the transactions need to be processed, for computing these attributes, in batch.

Converting computations on a batch of records to an equivalent incremental computation on individual records is an exercise akin to devising algorithms for incremental view maintenance. For example, for the telephone discount plan, there is a nontrivial mapping for

incrementally computing a persistent view for total expenses.

Figure 3 shows an illustrative embodiment of an implementation of the present invention. A workstation 800 comprised of a central processing unit 801 ("CPU") and storage 802. Storage 802 forms an efficient storage device providing fast access to stored data. Storage 802 comprises a main memory 803 and disk 804. The definitions of views, as well as the actual views themselves are stored in storage 802, along with relations that are not chronicles.

Data records for a chronicle are received sequentially through the communications port 805. The CPU 801 accesses storage device 802 to update persistent view stored therein based upon each new chronicle record received. The chronicle records themselves are typically not stored. However, if they are to be stored they are stored in a Slow Cheap Storage Device 806, which can be contained within workstation 800 (or be an external peripheral as shown in Fig. 3).

Queries or data processing commands are received by the CPU 801 via communication port 805 or pre-stored from storage 802. These queries are answered by the CPU after referring to the persistent views and other relations accessible from storage 802.

While the present invention has been described with respect to specific embodiments thereof, it will be understood by one skilled in the art that these are not exclusive embodiments. The present invention is applicable to numerous other application domains, including for example, credit cards, cellular telephone calls, stock trading, consumer banking, industrial control systems, retailing, frequent flier programs, etc., that do not depart from the spirit or principles of the present invention.

Claims

1. A method for using a computer system to process data of a transactional recording system database comprising the steps of:

defining a chronicle comprising a relation with a sequencing attribute, said relation comprising an unordered set of tuples;

defining other relations to capture a state of said system; and

generating a persistent view from said chronicle using a view definition language.

2. A method for using a computer system to process data of a transactional recording system database comprising the steps of:

incrementally maintaining a persistent view in response to transactions processed in said da-

tabase, said persistent view having been generated from a chronicle using a view definition language, said chronicle being defined to comprise a relation with a sequencing attribute, said relation comprising an unordered set of tuples; and

processing said data using said persistent view.

3. A method for using a computer system to answer queries of a transactional recording system database comprising the steps of:

defining a chronicle comprising a relation with a sequencing attribute, said relation comprising an unordered set of tuples;
defining other relations to capture a state of said system; and
generating a persistent view from said chronicle using a view definition language.

4. A method for using a computer system to answer queries of a transactional recording system database comprising the steps of:

incrementally maintaining a persistent view in response to transactions processed in said database, said persistent view having been generated from a chronicle using a view definition language, said chronicle being defined to comprise a relation with a sequencing attribute, said relation comprising an unordered set of tuples; and
answering said queries using said persistent view.

5. The method of claim 1 or 3 wherein the step of defining a chronicle comprises: marking an existing attribute of said relation as said sequencing attribute; or marking an additional attribute of said relation as said sequencing attribute.

6. The method of claim 1 or 3 wherein the step of generating a persistent view has a specific complexity for incremental computation.

7. The method of claim 1 or 3 wherein said view definition language is a chronicle algebra.

8. The method of claim 7 wherein said chronicle algebra comprises the steps of:

generating said chronicle by an intermediate chronicle algebra;

summarizing said chronicle produced by said intermediate chronicle algebra; and

mapping said chronicle produced by said intermediate chronicle algebra into said persistent view.

9. The method of claim 2 or 4 wherein said incremental maintenance of said persistent view comprises the steps of:

proactively updating a relation not in said chronicle;

said relation having a plurality of temporal versions, each one of said plurality for every said update;

joining said tuples of said chronicle with one of said plurality of said temporal versions of said relation;

updating said chronicle;

updating said persistent view in response to an append into said chronicle on which said persistent view depends; and

storing said updated persistent view.

10. The method of claim 2 or 4 wherein said step of incrementally maintaining said persistent view is in response to a single append into said chronicle in constant time; or a single append into said chronicle in time logarithmic in size of said relations; or a single append into said chronicle in time polynomial in size of said relations; or a single append into said chronicle in time polynomial in size of said chronicle.

11. The method of claim 1 or 3 wherein said step of generating persistent views comprises: selecting tuples of said chronicle which satisfy a predicate; or projecting said chronicle on attributes that include a sequencing attribute; or forming a natural equijoin between two said chronicles on said sequencing attribute; or

forming a union of two said chronicles in a same chronicle group, and having a same type; or determining a difference between two said chronicles in said same chronicle group, and having a same type; or deriving a groupby with aggregation, having grouping attributes, with a sequence number as one of said grouping attributes; or determining a cartesian product between said chronicle and any said relation, wherein an implicit temporal join on said sequencing attribute exists between said chronicle and said relation.

12. The method of claim 1 or 3 wherein a plurality of said persistent views are generated for said chronicle.
13. The method of claim 2 or 4 wherein a plurality of persistent views are incrementally maintained.
14. The method of claim 2 or 4 wherein said persistent view is maintained for a specified period and said persistent view is overwritten at end of said period.
15. A method for using a computer system to process queries of a transactional recording system database comprising the steps of:
- defining a chronicle comprising a relation with a sequencing attribute, said relation comprising an unordered set of tuples;
 - defining other relations to capture a state of said system;
 - generating a persistent view from said chronicle using a view definition language;
 - incrementally maintaining said persistent view in response to transactions processed in said database; and
 - answering said queries using said persistent view.
16. A method for using a computer system to process data of a transactional recording system database comprising the steps of:
- defining a chronicle comprising a relation with a sequencing attribute, said relation comprising an unordered set of tuples;
 - defining other relations to capture a state of said system;
 - generating a persistent view from said chronicle using a view definition language;
 - incrementally maintaining said persistent view in response to transactions processed in said database; and
 - processing said data using said persistent view.
17. A database system for processing data, comprising:
- means for defining a chronicle comprising a relation with a sequencing attribute, said relation
- comprising an unordered set of tuples;
- means for generating a persistent view from said chronicle using a view definition language;
- means for incrementally maintaining said persistent view in response to transactions processed in said database;
- means for updating said persistent view in response to an append into said chronicle on which said persistent view depends;
- means for storing said updated persistent view; and
- means for processing said data using said persistent view.
18. A database system for answering queries, comprising:
- means for defining a chronicle comprising a relation with a sequencing attribute, said relation comprising an unordered set of tuples;
 - means for generating a persistent view from said chronicle using a view definition language;
 - means for incrementally maintaining said persistent view in response to transactions processed in said database;
 - means for updating said persistent view in response to an append into said chronicle on which said persistent view depends;
 - means for storing said updated persistent view; and
 - means for answering said queries using said persistent view.
19. A database system for processing data, comprising:
- means for defining a chronicle comprising a relation with a sequencing attribute, said relation comprising an unordered set of tuples; and
 - means for generating a persistent view from said chronicle using a view definition language;
20. A database system for processing data, comprising:
- means for incrementally maintaining a persistent view in response to transactions processed

in said database, said persistent view having been generated from a chronicle using a view definition language, said chronicle being defined to comprise a relation with a sequencing attribute, said relation comprising an unordered set of tuples; 5

means for updating said persistent view in response to an append into said chronicle on which said persistent view depends; 10

means for storing said updated persistent view; and

means for processing said data using said persistent view. 15

21. A database system for answering queries, comprising:

20

means for defining a chronicle comprising a relation with a sequencing attribute, said relation comprising an unordered set of tuples; and

means for generating a persistent view from said chronicle using a view definition language. 25

22. A database system for answering queries, comprising:

30

means for incrementally maintaining a persistent view in response to transactions processed in said database, said persistent view having been generated from a chronicle using a view definition language, said chronicle being defined to comprise a relation with a sequencing attribute, said relation comprising an unordered set of tuples; 35

means for updating said persistent view in response to an append into said chronicle on which said persistent view depends; 40

means for storing said updated persistent view; and 45

means for answering said queries using said persistent view.

50

55

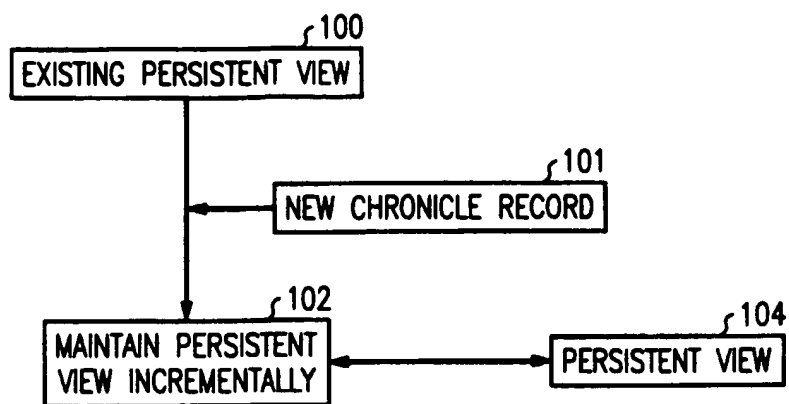


FIG. 1A

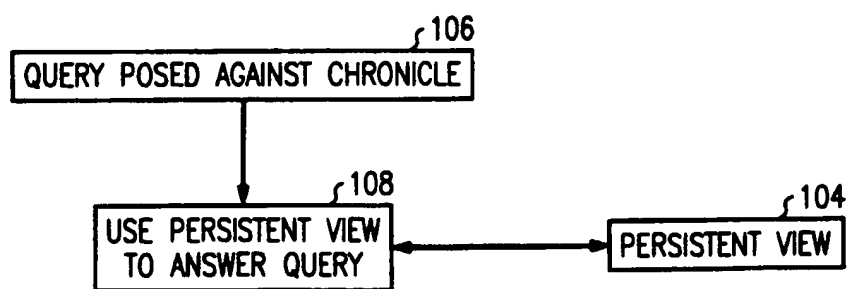


FIG. 1B

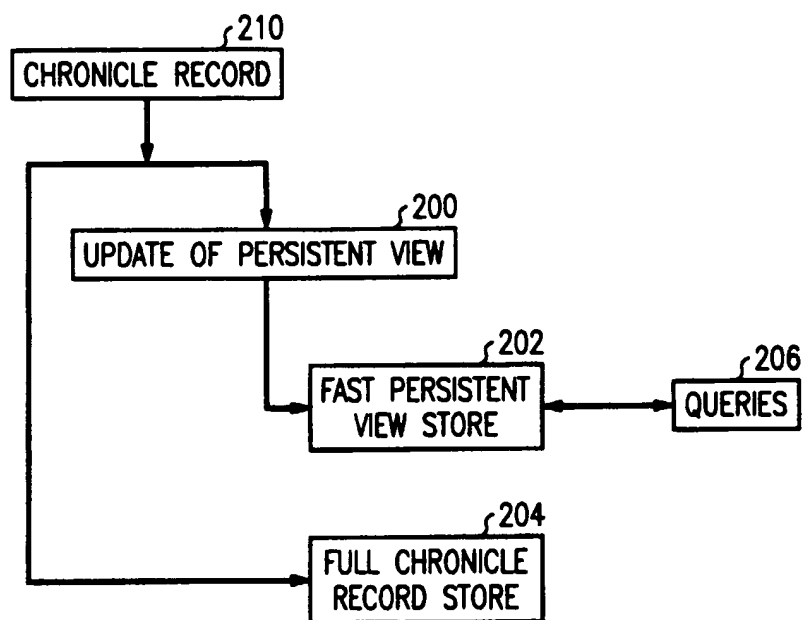


FIG. 2

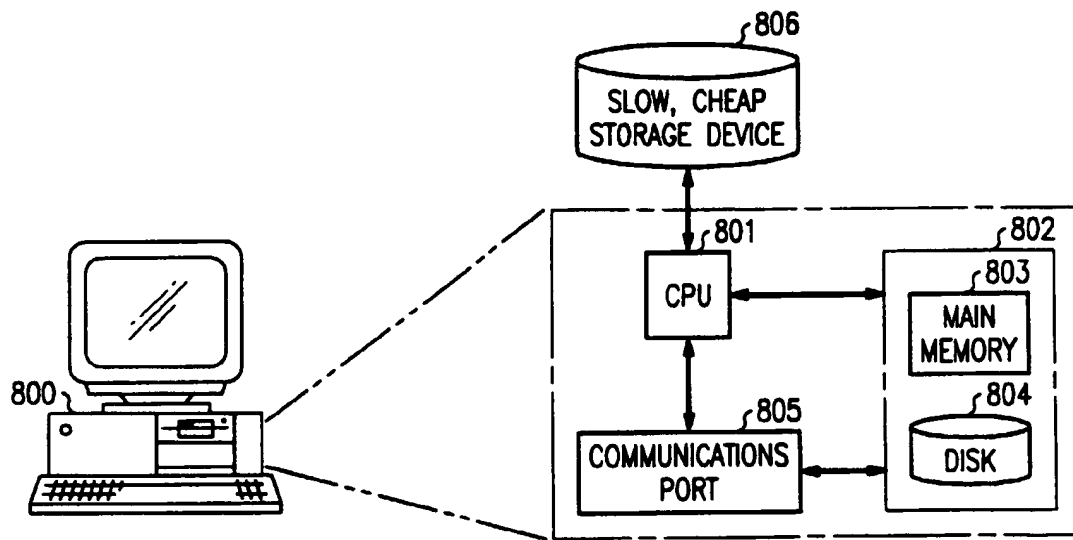


FIG. 3



(12) **EUROPEAN PATENT APPLICATION**

(43) Date of publication:
07.10.1998 Bulletin 1998/41

(51) Int. Cl.⁶: **G06F 17/30**

(21) Application number: **98103043.0**

(22) Date of filing: **20.02.1998**

(84) Designated Contracting States:
**AT BE CH DE DK ES FI FR GB GR IE IT LI LU MC
 NL PT SE**
 Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **01.04.1997 JP 83043/97**
29.05.1997 JP 139829/97

(71) Applicant: **Ogawa, Atsuro**
Nagoya-shi, Aichi, 460-0022 (JP)

(72) Inventor: **Ogawa, Atsuro**
Nagoya-shi, Aichi, 460-0022 (JP)

(74) Representative:
Sajda, Wolf E., Dipl.-Phys. et al
MEISSNER, BOLTE & PARTNER
Widenmayerstrasse 48
80538 München (DE)

(54) **Integrated database system and computer-readable recording medium recorded with program for managing database structure thereof**

(57) An integrated database system and a computer-readable recording medium on which a program is recorded for managing a database structure thereof which reduce word load and work cost by a relational database and a nonrelational database being integrated into a single system comprising storing means holding a database wherein a single record (10) is made up of a record part (10a) of a relational database and a record

part (10b) of a nonrelational database and the record part (10b) of the nonrelational database is made up of a parent record part (11) only or a parent record part (11) and at least one child record part (13) linked to the parent record part (11) and processing means for carrying out relational operation and data manipulation on the database.

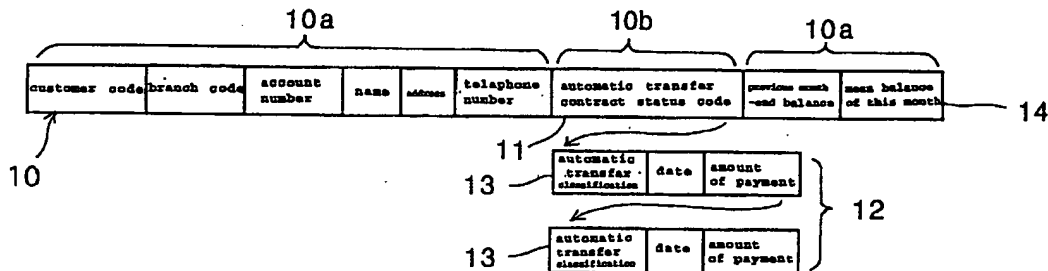


Fig. 1

Description

BACKGROUND OF THE INVENTION

This invention relates to a database wherein each record is made up of a record part of a relational database and a record part of a nonrelational database, and to a computer-readable recording medium recorded with a program for managing this database structure.

Conventionally, relational database systems and nonrelational database systems have been constructed as separate systems. Here, a relational database is a database made up only of data satisfying a normal form, and a nonrelational database is a database which is not limited to a normal form and is made up of either parent record parts only or parent record parts having one or more child record parts linked thereto. In systems having both a relational database system and a nonrelational database system of the configurations described above, data is converted in both directions between the relational database and the nonrelational database when data of the two systems is exchanged.

For example systems of financial institutions such as banks, life insurers, insurers and stockbrokerage companies have an off-line system using a relational database system and an on-line system using a nonrelational database system. In the on-line system, for example payment and receipt transactions inputted from terminals at branches are transmitted to a central facility and on the basis of these transactions data manipulation such as reference to and updating of the nonrelational database is carried out. The off-line system is operated for example from when the on-line business finishes, and on-line data is converted into off-line data to incorporate log information accumulated in the on-line system into the off-line relational database. Then, by performing relational operation and data manipulation processing on the off-line relational database, for example daily and monthly various statistical tables and documents are outputted and distributed to branches and return of information is thereby carried out. Also, by connecting this off-line relational database to a network, highly flexible non-fixed-form querying from branches and head office departments is made possible.

In a conventional normalized relational database, records are made up of relational lines only. That is, actual relational database systems have been developed with only the concept of 'relational line' = 'record'. Therefore, with respect to data for which this relational database structure is not ideal, for example in terms of processing efficiency or in terms of business processing, in practice inevitably a separate database system like the on-line system of a financial institution described above has had to be built and operated.

However, with the conventional configuration wherein a relational database and a nonrelational database are constructed as separate systems, because the

database is made up of two systems there has been the problem that the work of development, operation and maintenance and the like is complex. Also, because costs of development, operation and maintenance and so on are duplicated, there has been the problem that the system is expensive.

In the financial institution system described above, great expenditure is required for the development of the on-line system and for the development of the off-line system. Also, the operating hours of the on-line system and the off-line system are different and there is the problem that because the off-line business is started after the on-line business ends the overall operating time of the financial institution system is long.

Also, sometimes, when data manipulation such as updating and deleting is to be carried out on a predetermined data set of a nonrelational database constructed as a separate system, a relational operation is carried out on the relational database system to extract a predetermined record set from the relational database and then a data set is extracted from the nonrelational database on the basis of for example matching of keys with the extracted relational database records and the data manipulation of updating and deleting or the like is then carried out on the data set thus extracted from the nonrelational database. In this case, both the relational database and the nonrelational database must be read. Because the relational database and the nonrelational database constructed as separate systems are held in physically or logically separate drives, in a database system using a virtual space system, when both databases are read, paging occurs. Consequently, when both databases are read, paging occurs frequently and the throughput of the system falls.

From the point of view of the reliability of the database system also, non-matching of data between the databases readily occurs, and because there is generally a time lag between the two systems such non-matching of data is liable to go unnoticed, and this causes serious problems in business execution.

A data system disclosed in Unexamined Published Japanese Application No. S.62-209615 uses a non-normal form relational system including repeat items, but this system still presupposes a relational database and does not have the object of integrating a relational database and a nonrelational database.

SUMMARY OF THE INVENTION

It is therefore an object of the present invention to provide an integrated database system and a computer-readable recording medium recorded with a program for managing the database structure thereof which reduce work load and work cost by integrating a relational database and a nonrelational database into a single system.

In an integrated database system and a computer-readable recording medium recorded with a program for managing the database structure thereof provided by

the invention to achieve this object and other objects, each record is made up of a record part of a relational database and a record part of a nonrelational database, and relational operation and data manipulation are carried out on these records and as a result it is possible to provide a relational database and a nonrelational database as a single database. Therefore, the database system can be made a single system.

Also, the record part of the nonrelational database is made up of either a parent record part only or a parent record part and one or more child record parts linked to the parent record part. Therefore, in a database which for business processing reasons requires a parent record part and a child record part, it is not necessary to provide the two as separate tables.

In another integrated database system and computer-readable recording medium recorded with a program for managing the database structure thereof provided by the invention it is possible to process a relational operation on the record part of the relational database to extract a predetermined set of records and a data manipulation carried out on the extracted record set with a single command, and thus a relational operation and a data manipulation can be performed with a single reading.

With respect to data manipulation, raw data resulting from an execution in the nonrelational database part can be used in real time along with a relational database manipulation, and the reverse is also possible.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a view illustrating a record structure of a preferred embodiment of the invention;

Fig. 2 is a schematic view showing the structure of a single record in the preferred embodiment;

Fig. 3 is a view illustrating a linking structure of variable parts; and

Fig. 4 is a schematic view showing a database structure of the preferred embodiment.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

A preferred embodiment of the invention will now be described with reference to the accompanying drawings.

A database system of a preferred embodiment of the invention is made up of a magnetic disc serving as storing means holding a database, a database management program serving as processing means loaded into real memory, and a CPU serving as a processing device for executing the database management program and so on. The database management program manages a database structure of this preferred embodiment which will be described below and carries out relational operation and data manipulation on the database.

A record of the database of this preferred embodi-

ment is shown in Fig. 1. A single record 10 of this preferred embodiment is an example of a record configuration of the invention applied to for example a database of a banking system, and is made up of a record part of a relational database (hereinafter, 'record part of a relational database' will be abbreviated to 'relational record part') 10a and a record part of a nonrelational database (hereinafter, 'record part of a nonrelational database' will be abbreviated to 'nonrelational record part') 10b. The relational record part 10a shows customer information and the nonrelational record part 10b shows customer account information. The nonrelational record part 10b generally includes an account balance and a bankbook balance and so on, but for brevity the specific details thereof will not be described here.

The relational record part 10a is for example made up of fields for a customer code, a branch code, an account number, a name, an address, a telephone number, a previous month-end balance and a mean balance of this month.

The nonrelational record part 10b is made up of a parent record part 11 and a child record part 12. In the example shown in Fig. 1, the parent record part 11 is made up of a single parent record having a field for an automatic transfer contract status code, but it is of course also possible for the parent record part to be made up of a plurality of parent records. A pointer pointing to the first child record 13 of a plurality of child records 13 constituting the child record part 12 is held in the automatic transfer contract status code field. The child records 13 are each made up of data fields for an automatic transfer classification, a date and a payment amount, and a pointer field for pointing to the next child record 13. No particular limit is set on the number of child records 13 making up the child record part 12, and the child record part 12 as a whole is made of variable length.

The fields included in each of the child records 13 are in practice fields for a classification such as electricity, water, loan, check, bill and so on of an automatic transfer payment automatically debited from the account and for the date and the amount of each payment.

The fields of the relational record part 10a can become object fields of relational operations such as conditional searches and of data manipulations such as referencing, updating, inserting, and deleting. Here, by means of a relational operation such as a conditional search, in addition to extracting a specified record group from a single table, it is also possible to extract a specified record group from a plurality of tables. Data manipulation alone, not accompanying a relational operation, may be carried out on the nonrelational record part 10b, and also the fields of the parent record part thereof may be treated as object fields of a relational operation. Also, if a single child record 13 of the child record part 12 of the nonrelational record part 10b is specified and a sin-

gle field is selected from among a plurality of fields of that child record 13, that selected field can be made an object of a relational operation.

The record 10 can also be thought of as being made up of a child record part 12 of variable length linked by pointers and a basic part 14 generally of a fixed length.

The structure of a general record 100 of this preferred embodiment is shown schematically in Fig. 2. The record 100 is made up of a relational record part 101, a parent record part 103 of a nonrelational record part 102, and two child record parts 104a, 104b linked to the parent record part 103. The record 100 can also be thought of as a variable part 104 of variable length made up of child record parts 104a, 104b each of variable length and a basic part 105 of a fixed length. In Fig. 2, the relational record part 101 and the nonrelational record part 102 are each shown as making up an integrated field set in the basic part 105, but the fields making up the relational record part 101 and the nonrelational record part 102 may alternatively be out of order and mixed in the basic part 105.

The child record fields each have a pointer pointing to the next child record, and are linked in one direction. The pointer pointing to the next child record is made up of the page number of the page where the next child record is and for example the offset of the next child record in that page. The offset shows a position from the beginning of the page. An EOF (End of Field) marker is set in the pointer held in the field of the last child record of the child record part. In Fig. 2, two child record parts 104a, 104b are each linked to the parent record part 103, but alternatively the number of child record parts linked to the parent record part 103 may be zero. That is, the nonrelational record part 102 may be made up of the parent record part 103 only.

The location of the record 100 will now be explained using Fig. 3, taking the child record part 104a as an example. The basic part 105 of the record 100 is held in a basic area 110a of a page 110, and the child record part of the record 100 is held in a free area 110b of the page 110. The pages 110 are delimited into for example 4K units. Although in Fig. 3 an example wherein the child record part spreads over a plurality of pages is shown, by the child records constituting the child record part being disposed close together it is possible for a single record 100 to be held in a single page. By this means it is possible to reduce the rate of incidence of paging.

An example of a database 120 of the present preferred embodiment made up of multiple records is shown in Fig. 4. Because as many variable parts 121 extending in a row direction and a line direction are generated as the necessary number of child records, the database 120 can be prevented from occupying unnecessary area. Also, since a two-dimensional table made up of basic parts 105 of a fixed length is normalized, the database 120 of this preferred embodiment shown in

Fig. 4 made up of this normalized two-dimensional table and the variable parts 121 can be processed highly efficiently like a conventional normalized relational database. Furthermore, because in addition to the two-dimensional table made up of the basic parts 105 the database 120 has a variable number of child record groups formed in the direction of a third dimension by the variable parts 121, it is not necessary for the two-dimensional table part and the child record groups of variable number to be made separate tables. Therefore, the input-output load on the database is reduced.

Next, referencing and updating of the data in this preferred embodiment, and data manipulations consisting of the addition and deletion of child records and so on will now be described on the basis of Fig. 3.

(1) Referencing and Updating

- 1) Since each record has a unique record number (RID), using a specified RID the in-page offset of that record is obtained from the page ID maps by a known method and the record is found.
- 2) A target field is found from the in-record offset of that field, and the field type is compared with the data type of the actual data as a check. In the case of a child record, the target child record is found from the child record number by following the pointers.
- 3) In the case of referencing, data is read from the field or child record, and in the case of updating, data is written into the field or child record.

(2) Adding a Child Record

- 1) The page where an immediately preceding child record to which a new child record is to be connected lies is found and by a known method the map page is searched to check how much free space there is on that page. When there is no free space, a free space in the following page is found.
- 2) When a page having free space is found, the size of the free space and the size of the child record are compared and if necessary the page is condensed and if the addition can be made the in-page offset of the free space is obtained and for example the respective RID and in-page offset are set in the pointer of the above-mentioned immediately preceding child record. Then, the pointer value hitherto had by the immediately preceding child record is set in the pointer area of the child record being added and these are written into the free space and if necessary the page free space state shown by the map page is updated by a known method.
- 3) When the free space is smaller than the size of the child record even after the page is condensed, another page with free space is found from the map page.

(3) Deleting a Child Record

- 1) The page including the child record immediately preceding the child record to be deleted is found and in the pointer area of that child record is set the pointer value had by the child record to be deleted.
- 2) A deletion flag of the child record to be deleted is turned on. This child record is actually deleted when the page is next condensed.
- 3) After the deletion flag is turned on in 2), if necessary the page free space state shown by the map page is updated by a known method.

(4) Batch Manipulation of Child Records

The child records are connected by pointers and lie in nearby pages. Therefore, it is possible to refer to, update, add and delete a chain of child records efficiently in a batch. Batch manipulation is carried out by following the pointers and repeating the processes (1), (2) or (3) described above.

The extraction of data manipulation object records described above may be carried out using a designated RID or a predetermined record set may be extracted by performing a relational operation on the relational record part and then data manipulation carried out on at least either the relational record part or the nonrelational record part of the extracted record set. In this case, the relational operation and the data manipulation can be processed with a single command.

In the preferred embodiment of the present invention described above, by the relational record part 10a and the nonrelational record part 10b being made to constitute a single record 10, it is possible to provide an off-line system and an on-line system of a financial institution system as a single database system. Therefore, because development, operation and maintenance become those of a single system the work becomes easy and the work load is reduced. Also, the cost of development, operation and maintenance decreases.

Furthermore, with a single command it is possible to execute processing to carry out a relational operation on the relational record part 10a to extract a predetermined record set and perform a data manipulation on at least either the relational record part 10a or the nonrelational record part 10b of this extracted record set. As a result, the problem of paging occurring and processing efficiency consequently falling when a relational database and a nonrelational database are both referenced in a virtual space system because the two databases are constructed as separate systems does not arise.

Although in the preferred embodiment described above a single nonrelational record part 10b was combined with a single relational record part 10a, it is also possible to combine a plurality of nonrelational record parts with a single relational record part.

Also, in the preferred embodiment described above it is possible to make up the nonrelational record part

10b with parent record parts only. That is, it is possible to make up a nonrelational record part with parent record parts only which could not have existed in a conventional nonrelational database alone.

Also, when the fields inside each child record of the child record part are limited to a single data item only, a logical structure wherein non-normal-form repeated data elements of a relational database are disposed on a third dimension side is obtained. Therefore, because on the two-dimensional space side a normal form of a relational database can be secured, highly efficient handling becomes possible.

Also, in the preferred embodiment described above, because only the necessary number of child records are provided in the child record part, empty spaces are not formed in the database and the area occupied by the database can be reduced.

A database management program for managing a database structure of this invention can be provided recorded on a CD-ROM, FD, magnetic tape or other recording medium. It can also be provided over a network such as the Internet.

With the present invention, because it is possible to provide a relational database and a nonrelational database as a single database, a database system can be integrated into a single system. Therefore, the work of developing, operating and maintaining the system becomes easy and the work load decreases. Furthermore, the cost of development, operation and maintenance of the system decreases.

Also, it is possible to enjoy a high processing efficiency obtained through the freedom of record structure allowed by a conventional nonrelational database without losing the degree of freedom of querying allowed by a normalized conventional relational database.

Furthermore, in this invention, raw data executed in the nonrelational database part can be connected to a relational database manipulation in real time and for example immediately reflected into a data manipulation such as a query. In this case also, the generally known effects of a normalized database can be fully obtained in the relational database part.

Also, because the nonrelational data and the relational data of a record can be checked against each other one record at a time, non-matching between the nonrelational data and the relational data does not readily occur. Furthermore, since duplicated data can be integrated, non-matching can be prevented.

Also, because in the present invention it is not necessary to put parent records and child records in different tables, the processing efficiency of data manipulation rises. That is, in the environment of a virtual space system, with respect to parts held in the same page the input-output load is reduced. Also, if a relational operation on a relational record part and data manipulation carried out on at least either a relational record part or a nonrelational record part are processed with a single command, since the number of readings of

the database can be decreased and in a virtual space system the incidence of paging can be minimized, the throughput of the system can be increased. Also, because a child record part can be realized as a record part conceptually in a third dimension direction, it is possible to avoid user confusion in handling the data. It is notable in this connection that in relational databases expression on the basis of the concept of a two-dimensional table or combinations thereof is carried out commonly.

the extracted set of records (10, 100).

Claims

1. An integrated database system comprising:

storing means holding a database wherein a single record (10, 100) is made up of a record part (10a, 101) of a relational database and a record part (10b, 102) of a nonrelational database and the record part (10b, 102) of the non-relational database is made up of a parent record part (11, 103) only or a parent record part (11, 103) and at least one child record part (13, 104) linked to the parent record part (11, 103); and processing means for carrying out relational operation and data manipulation on the database.

2. The system according to claim 1, wherein it is possible with a single command to process a relational operation with the record part (10a, 101) of the relational database as an object of operation to extract a predetermined set of records (10, 100) and a data manipulation carried out on the extracted set of records (10, 100).
3. A computer-readable recording medium on which a program is recorded for performing relational operation and data manipulation on a database structure, wherein a record (10, 100) is made up of a record part (10a, 101) of a relational database and a record part (10b, 102) of a nonrelational database and wherein the record part (10b, 102) of the non-relational database is made up of a parent record part (11, 103) only or a parent record part (11, 103) and at least one child record part (13, 104) linked to the parent record part (11, 103).
4. The computer-readable recording medium on which a program is recorded for managing a database structure according to claim 3, wherein it is possible with a single command to process a relational operation with the record part (10a, 101) of the relational database as an object of operation to extract a predetermined set of records (10, 100) and a data manipulation carried out on

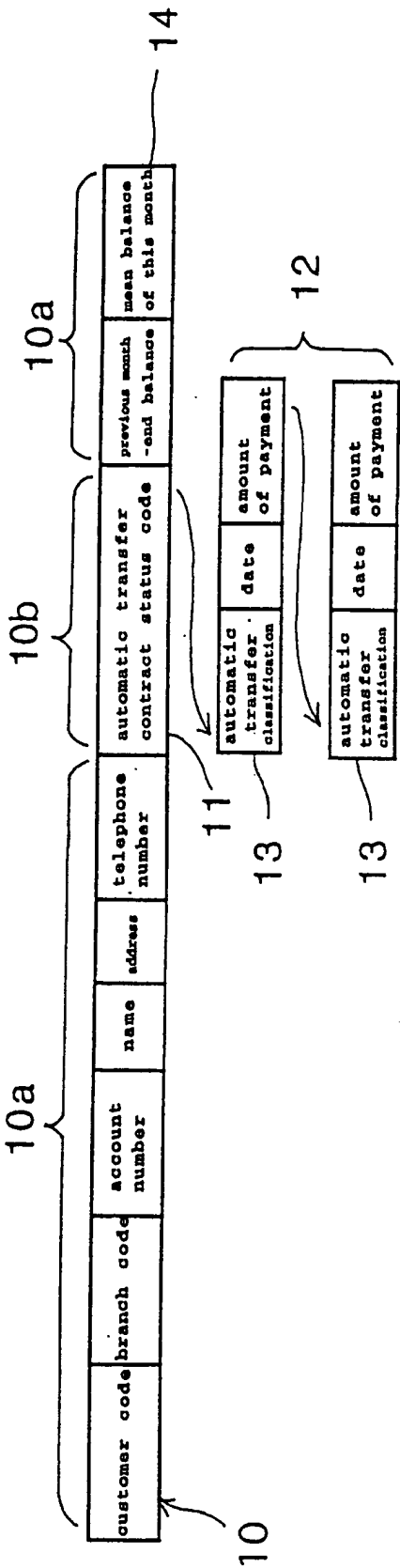


Fig. 1

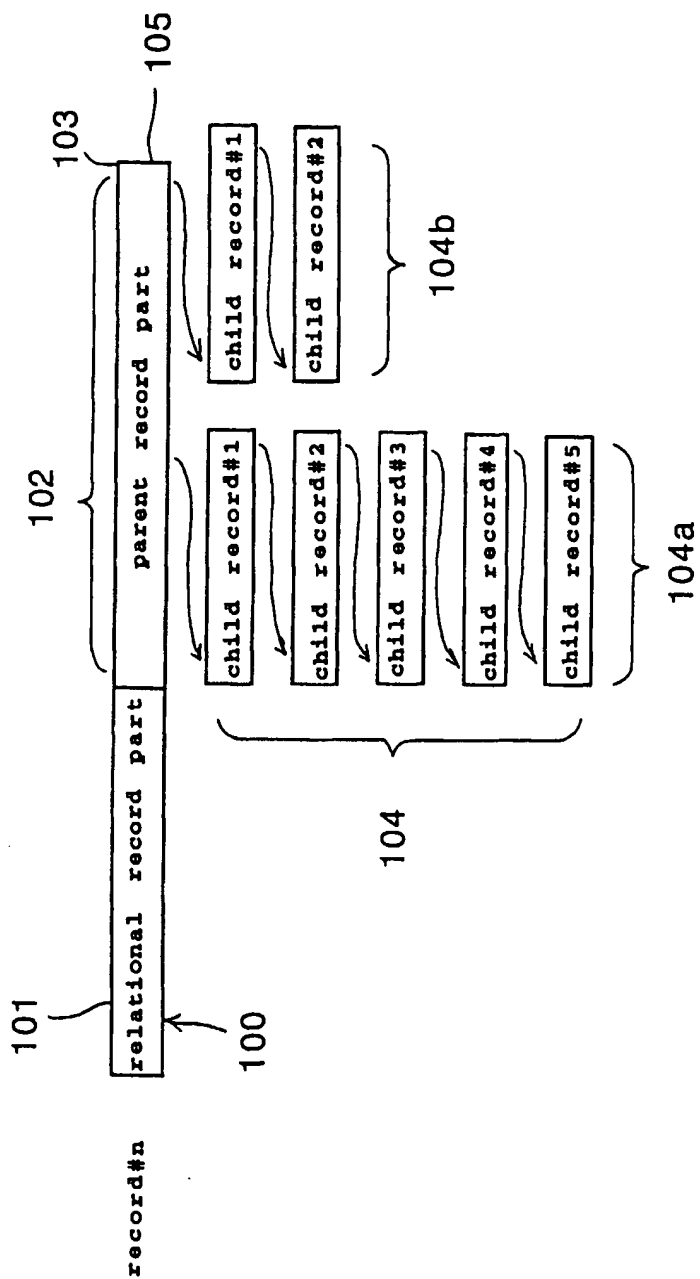
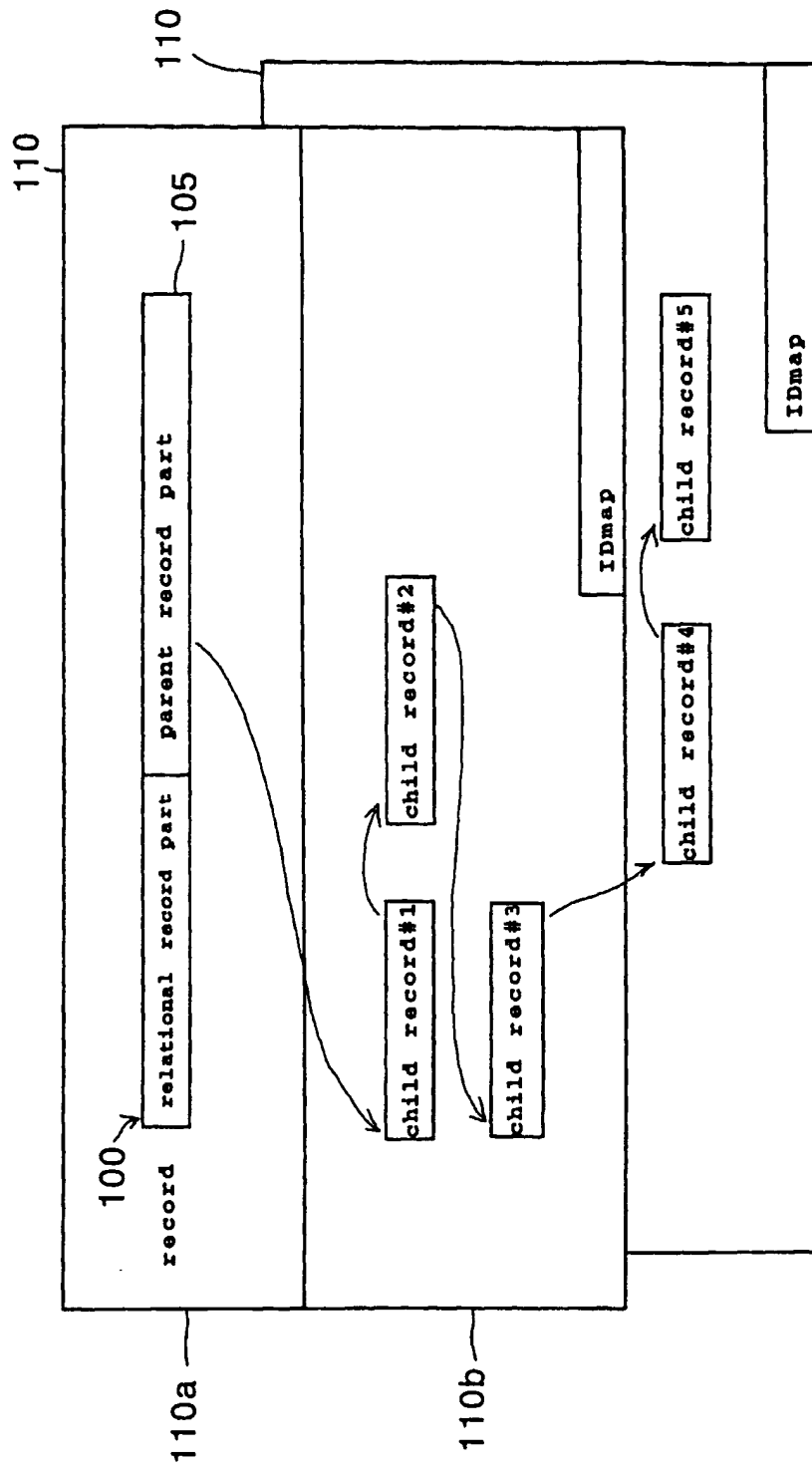


Fig. 2

Fig. 3



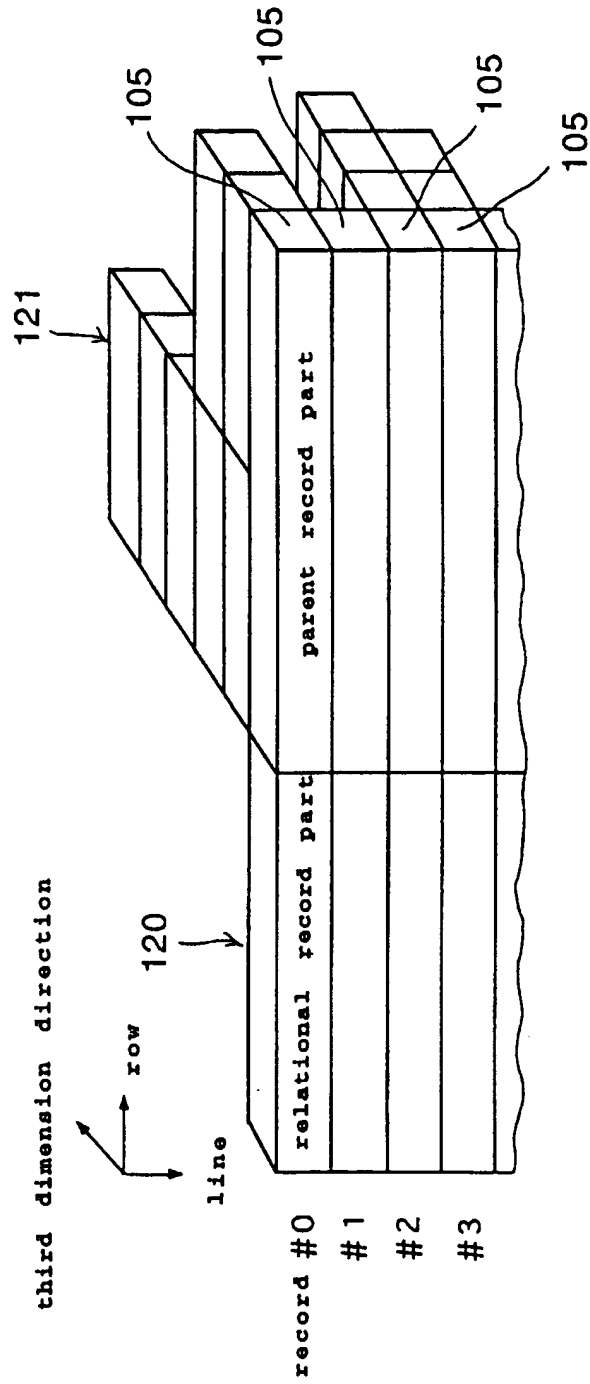


Fig. 4

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁶ : G06F 3/00, 13/00		A1	(11) International Publication Number: WO 95/08794
			(43) International Publication Date: 30 March 1995 (30.03.95)
(21) International Application Number: PCT/US94/08934			(81) Designated States: AU, CA, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).
(22) International Filing Date: 8 August 1994 (08.08.94)			
(30) Priority Data: 08/123,657 20 September 1993 (20.09.93) US			
(71) Applicant: CODEX, INC., a subsidiary company of MOTOROLA INC. [US/US]; 20 Cabot Boulevard, Mansfield, MA 02048 (US).			
(72) Inventors: GILBERT, Yuval; 15A Auburn Street, Brookline, MA 02146 (US). ST. PIERRE, Edgar; 1510 Highland Avenue, Fall River, MA 02720 (US). HODGSON, Brian; 17 Jackson Terrace, Newton, MA 02158 (US). BARBELL, Benita; 95 Pond Street, Sharon, MA 02067 (US).			
(74) Agents: PARMELEE, Steven, G. et al.; Motorola Inc., Intellectual Property Dept./JRW, 1303 East Algonquin Road, Schaumburg, IL 60196 (US).			

Published

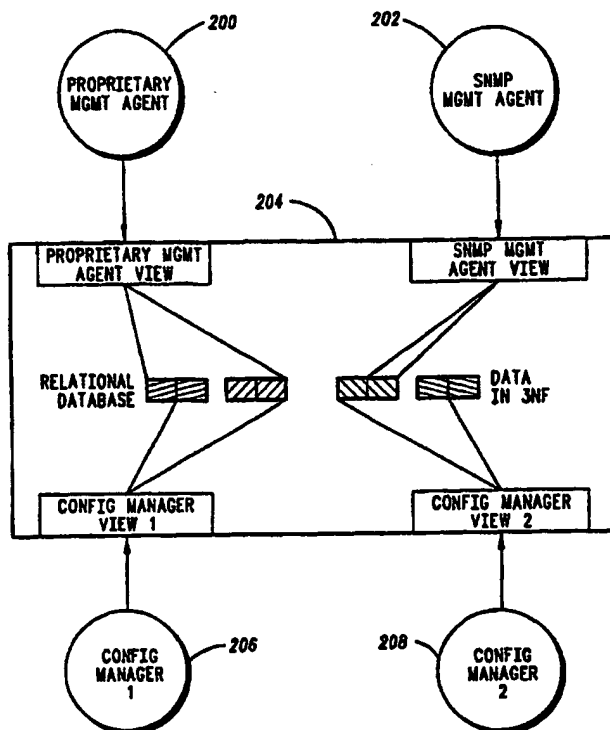
With international search report.

Before the expiration of the time limits for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: A MANAGEMENT AGENT SYSTEM FOR THE SUPPORT OF MULTIPLE NETWORK MANAGERS, AND A METHOD FOR OPERATING SUCH A SYSTEM

(57) Abstract

A communication system allows for the separation between the logical, external view of a network manager (10, 12) and the physical, internal view of the communication device (18, 20). A single management agent (108, 110) represents a single network manager (100, 102). The management agent (108, 110) is responsible for interpreting requests from that network manager (100, 102).



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	GB	United Kingdom	MR	Mauritania
AU	Australia	GE	Georgia	MW	Malawi
BB	Barbados	GN	Guinea	NE	Niger
BE	Belgium	GR	Greece	NL	Netherlands
BF	Burkina Faso	HU	Hungary	NO	Norway
BG	Bulgaria	IE	Ireland	NZ	New Zealand
BJ	Benin	IT	Italy	PL	Poland
BR	Brazil	JP	Japan	PT	Portugal
BY	Belarus	KE	Kenya	RO	Romania
CA	Canada	KG	Kyrgyzstan	RU	Russian Federation
CF	Central African Republic	KP	Democratic People's Republic of Korea	SD	Sudan
CG	Congo	KR	Republic of Korea	SE	Sweden
CH	Switzerland	KZ	Kazakhstan	SI	Slovenia
CI	Côte d'Ivoire	LI	Liechtenstein	SK	Slovakia
CM	Cameroon	LK	Sri Lanka	SN	Senegal
CN	China	LU	Luxembourg	TD	Chad
CS	Czechoslovakia	LV	Latvia	TG	Togo
CZ	Czech Republic	MC	Monaco	TJ	Tajikistan
DE	Germany	MD	Republic of Moldova	TT	Trinidad and Tobago
DK	Denmark	MG	Madagascar	UA	Ukraine
ES	Spain	ML	Mali	US	United States of America
FI	Finland	MN	Mongolia	UZ	Uzbekistan
FR	France			VN	Viet Nam
GA	Gabon				

A MANAGEMENT AGENT SYSTEM FOR THE SUPPORT OF MULTIPLE NETWORK MANAGERS, AND A METHOD FOR OPERATING SUCH A SYSTEM

5

Field of the Invention

This invention relates generally to communications networks, and specifically to the management of a communication network.

10

Background of the Invention

A communication network consists of a number of users communicating through various inter-connected communication devices. The communication devices could be terminals, host computers, multiplexers, frame handlers, etc.

15

Such a network may serve a large number of users. The management of such a network may be difficult and complex. For example, if one user desires to transfer information to another user in the network, the network must determine the most efficient path from one user to the other, determine the types of devices used to interface the user to the network, and configure the path so that the information will flow efficiently between the users.

20

25

A communication device may have a number of "views". A view is a logical grouping of one or (typically) more data items together to form a logical abstraction that represents a higher level concept for that group of parameters. For example, one view of a "port" may be that it is made up of 14 particular parameters (timers, lead values, etc). A different view of that same port may be that it is made up of 10 of those parameters plus 2 different parameters. The view may change depending on the information needed to configure or monitor the port.

30

35

When the ability to manage multiple views is not available, then all network managers on a network must share the same view. If the multiple network managers can not
5 share the same view, then multiple versions of each configuration manager on the communications device must be produced (one for each network manager), or each management agent in the communications device must maintain a complete understanding of the system view of the communication
10 cevice. Either alternative adds to the amount of memory used by the device which adds to the cost and/or reduces the available throughput of the device. Both alternatives also add significantly to the maintenance cost of the software.

15 A system that solves the problem of supporting multiple, diverse network managers by providing a means to translate multiple management views into the normalized, internal view of the communication device would thus be valuable.

20 Brief Description of the Drawings

FIG. 1 is a customer network.

25 FIG. 2 is a management agent system for the network/communication device.

FIG. 3 is a block diagram of the multiple views supported within the communication device.

30 FIG. 4 is a flow chart showing the handling of a management request by the management agent system.

Description of the Preferred Embodiment

A system is shown that allows for the separation between the logical, external view of a network manager and the physical, internal view of the communication device. A single management agent represents a single network manager and is responsible for interpreting requests from that network manager.

10 A Third Normal Form (3NF) data model of the communication device, embodied in base tables within a relational database, allow the communication device management agent - on behalf of the network manager - and communication device's configuration managers to map each other's specific data to this normalized view of the communication device.

This management agent system is thus easily adaptable to a communication device's changing requirements and complex data modeling environment, reducing the cost of building and maintaining the system.

FIG. 1 shows a network consisting of multiple communication devices 14, 16, 18, 20 interconnected by public or private leased lines through the Public Switched Telephone Network (PSTN) 22. Communication devices 14, 16, 18, 20 may be terminals, host computers, multiplexers, frame handlers, etc. Communication devices 14, 16, 18, 20 transport customer voice and data traffic through the network. Attached to the communication devices at the edges of the customer network is a proprietary network manager 10. Proprietary network manager 10 configures and monitors the communication devices 14, 16, 18, 20. The Simple Network Management Protocol (SNMP) Manager 12 is coupled with the network through communication device 16. The SNMP Manager

is also used to configure and monitor communication devices 14, 16, 18, 20.

FIG. 2 shows the elements of the management agent system. Proprietary network manager 100 is connected to communication device 104. SNMP Manager 102 is also connected to communication device 104.

Internal to the communication device 104, a proprietary management agent 108 provides agent functionality within communication device 104 for proprietary network manager 100 by using the protocol and semantics that are native to network manager 100. Likewise, proprietary management agent 110 provides agent functionality within communication device 104 for SNMP manager 102 by using the protocol and semantics that are native to SNMP manager 102.

The protocol and semantics native to each network manager are defined as a set of managed objects. A "managed object" is a "view", as defined earlier, specifically defined by each network manager. A managed object view may be made up of data attributes, operations and behavior (which describes what the abstract "view" such as a port will do under different conditions).

25

For example, proprietary network manager 100 may send an operation to communications device 104 to set the output lead of a particular port to a specific voltage level. The lead itself would be defined as a data attribute of the port managed object. The behavior would be that when that attribute is set to a particular value, then the output lead is changed to a particular voltage level. The operation, the attribute and the behavior are all part of that managed object "view". A different "view" of that same port from the SNMP manager may

30

require a different attribute and operation to achieve the same result.

Management agents 108 and 110 also maintain limited modeling knowledge of communication device 104, including knowledge of the database schema and how managed objects are configured together. The database schema is the set of logical relationships and organization applied to the data in the database. The database maintains a normalized (third normal form - 3NF) representation of the configuration data for communications device 104. This is a "view" of the data which is commonly understood by all agents and configuration managers that reference the database.

Management agents 108,110 translate the network manager managed object "view" into the communication device normalized "view" in the database.

A request by network manager 100, 102 can be broken down into the following categories:

- Create Configuration--identifies a new instance of a managed object "view". Since there are many ports on communication device 104, each individual port is identified by an individual instance. Creating a new instance makes it available for configuration and monitoring.
- Delete Configuration--removes an instance of a managed object "view" for the list of instances available for configuration and monitoring.
- Read Configuration--query particular data attributes from a particular instance of a managed object "view"

6

- **Modify Configuration**--alters the value of a particular data attribute in a particular instance of a managed object "view"

5 • **Perform an Action** (such as activate a call)--executes particular operations on an instance of a managed object "view" that can not otherwise be performed by modifying the configuration.

10 Each management agent 108, 110 is also able to send messages to its respective network manager 100, 102 whenever significant events occur on communications device 104.

15 Management agents 108, 110 interpret the request from its respective network manager 100, 102, by translating it into a database read or write request. Configuration is validated at the management agent 108, 110, resources are validated and reserved, and then all the data written to the
20 database. Management agent 108, 110 then notifies the appropriate communication device configuration managers 114, 116. Communication managers 114, 116 implement the request. Management agent 108, 110 then formulates an appropriate response message back to the respective network
25 manager 100, 102.

Validation of a request by management agents 108, 110 are performed within the context of the managed object. Each management agent 108, 110 performs validation of network
30 manager requests to the extent that the request can be accepted into the database. This includes checking per parameter validation, such as checking data attribute boundaries (within minimum/maximum values allowed for the attribute); intra-managed object relationships, such as
35 whether one attribute will work with another attribute set to

a particular value; and inter-managed object validation, which may be the same as inter-managed object validation, but the attributes that are compared may be in different managed object "views". The validation is concerned primarily with the configuration definition, and is performed within the context of the current nodal configuration. It does not account for the instantiation of the configuration.

Not only does the management agent embody the network management view, it also has a limited understanding of the system view of communication device 104. It must know when a request is dependent on the behavior of multiple configuration managers, and how the request must be coordinated between those configuration managers 114, 116.

15

Each management agent 108, 110 performs the following tasks:

- Translation from network manager managed object "view" into communication device normalized "view"
- Validation of network manager requests as it pertains to the managed object "views".
- Physical resource validation & reservation (such as ports).
- Storing configuration data in the database.
- Sending notifications to the appropriate communication device configuration managers regarding changes to the configuration database.
- Translate and route managed object-specific actions to the appropriate communication device configuration manager.

30
35

- Translate and route event messages from the communication device configuration managers to the network manager.

5

Database server 112 (DBS) stores configuration data in a relational form, and thereby allows multiple views of data. DBS may be composed of a memory, such as random access memory and disk storage. DBS 112 can support having two
10 databases open at one time.

The database of DBS 112 provides a unified, normalized approach for managing configuration data, regardless of how the network managers and configuration managers "view" the
15 data. The abstraction of communication device 104 is captured in the database, where data can be accessed by specifying appropriate criteria. Direct users of DBS 112 (e.g., management agents or configuration managers) translate their private "views" into and out of the normalized tables that make up the
20 database. These direct users then use the database to create, delete, query, update, and search these tables according to varying criteria. The database contains all nodal configuration data.

25 When notified of configuration changes to the on-line configuration, configuration managers 114, 116, read those changes from DBS 112, and initiate their implementation in the node.

30 Configuration managers 114, 116 implement configuration requests within communication device 104 along functional boundaries. Configuration managers 114, 116 know about a single, on-line database which it uses to retrieve configuration data. Configuration managers 114, 116
35 implement the request by generating their own requests to the

entity or entities to be configured, such as port 118 or call 120. Port 118 may be an EIA232 port connected to a host computer, or a T1 port connected to frame handler. Call 120 may be a connection through the communications network from one EIA232 port to another EIA232 port on another communications device.

Configuration managers 114, 116 read the database via DBS 112 to obtain configuration updates and initialization data. When updates have been made, configuration managers 114, 116 read the update so that it can instantiate the change. Configuration managers 114, 116 also read the database via DBS 112 during node initialization.

The operations between either management agent 108, 110 and either or both configuration managers 114, 116 is specified as part of the configuration manager interface specification. There are a generic set of operations that are available on all configuration managers 114, 116. These operations are the same as, and follow the same definitions given earlier for Create, Delete and Modify.

FIG. 3 shows the support of multiple views by the database. One of the functions of the relational database is to provide the ability for its multiple users (clients) to maintain separate and distinct view of the data in the database. The view of proprietary network manager 100 (FIG. 2) view is embodied in a Proprietary management agent (200). Similarly, the view of SNMP manager 102 (FIG. 2) is embodied by SNMP management agent 202. A relational database 204 contains base tables defined in 3NF. Relational database 204 contains the implementation of the data model of communication device 104 (FIG. 2) and the implementation view of communication device 104 as embodied by configuration managers 114, 116.

Database 204 allows support of multiple versions of configuration data: on-line (or active), off-line (or alternate), etc. For example, "off-line" configuration data can be specified, validated, and stored in database 204 without
5 requiring communication device 104 to implement such configuration. Communication device 104 can be subsequently rebooted to use this off-line database. Only one configuration is "on-line" or active at any particular time.

10 FIG. 4 is a flow chart for the handling of management requests by the management agent system shown in FIGs. 2 and 3. Proprietary network manager 100 sends a management request to communication device 104 to configure a new port 118 (Step 302). The management request is received by
15 communication device 104 and routed to the appropriate management agent, in this case proprietary management agent 108 (Step 304). The management agent 108 validates the port configuration request for syntactic and semantic correctness (Step 306). If the request is not valid (Step 308), the
20 management agent 108 returns an error response to the network manager 100 (Step 310). If the management request is valid, the management agent 108 maps the network manager-organized configuration data for the port into the 3NF normalized form. (Step 312). Management agent 108 updates
25 the base tables containing port configuration using the database server 112 (Step 314). After the database is updated, management agent 108 notifies configuration manager 114 that there is a new port configuration data in the database (Step 316). Configuration manager 114 reads the new port
30 configuration data in 3NF form using the services of the database server 112 (Step 318). Configuration manager 114 maps the data from 3NF form into the communication device 114 specific form required to implement the port's configuration (Step 320). The configuration manager 114
35 implements the port configuration by talking to the port 118

and providing it the configuration data (Step 322). The configuration manager 114 responds to the management agent 108 that the port has been configured (Step 324). The management agent 108 sends a management response back to the network manager 100 indicating the management request was successfully completed (Step 326). Finally, the network manager 100 receives the management response from the management agent 108 (Step 328).

10

Conclusion

This combination of the following elements relational network managers 100, 102, management agents 108, 100 and database 204 storing information in (3NF) isolates the external management view from the internal physical view by the use of data mapping is new. This scheme allows for the easy addition of new "views" (e.g. management agents), which are independent from existing views, and therefore have no impact to the existing views.

20

Because each view of communication device 104 has its own model for looking at its data, these views (external models) are mapped onto a "normalized" data model which is in 3NF form. The internal views (e.g., configuration managers 114, 116) also map to the database #nf view. This allows for multiple interpretations of the management model by different portions of the communication device.

Since external modeling of a communication device can often be orthogonal to the actual internal implementation, a means to reduce the complexity of this mapping is essential. Doing this mapping using data (via a database) instead of code is the means offered in this invention.

Thus, the described system supports the capability to provide multiple views easily. This management agent system is easily adaptable to a communication device's changing requirements and complex data modeling environment. Further,
5 the complexity of mapping between network manager and the communication device is reduced.

The system also avoids imposing the network management/user view on the internal communication device,
10 and avoids imposing internal implementations within the communication device on the network management/user view. This avoids having changes in one view affect the other view, resulting in greater efficiency and adaptability.

15 The network manager/user view is hidden from internal implementations, allowing either to change as required by the network. Further, the addition of multiple network managers may be easily accomplished by adding new management agents.

20 All of these advantages add up to reduced development and maintenance costs for communications device 104, as well as reduced random access memory utilization in the device, thereby reducing its cost to manufacture.

We Claim:

1. A management agent system for a communication device comprising:
 - 5 a network manager for configuring the communication device, the network manager having configuration information for configuring the communication device;
 - 10 a communication device management agent, coupled to the network manager, where the communication device management agent handles management requests on behalf of the network manager;
 - 15 a communication device database, coupled to the communication device management agent, where the communication device management agent stores configuration information for the communication device;
 - 20 a communication device configuration manager, coupled to the communication device database and the communication device, the communication device configuration manager reading the configuration information from the communication device database, and then configuring the communication device.
 - 25
2. The management agent system of claim 1 where there are a plurality of network managers.
3. The management agent system of claim 2 where the
 - 30 communication device management agent maps the communication device information into the communication device database.
4. The management agent system of claim 3 where the
 - 35 mapping of the communication device information into the

communication device database converts the communication device information into a third normal form.

5 5. The management agent system of claim 4 where at least two of the network managers have different organizational views of the communication device information.

10 6. The management agent system of claim 5 where the communication device configuration manager converts the communication device information stored in the communication device database from third normal form into information for configuring the communication device.

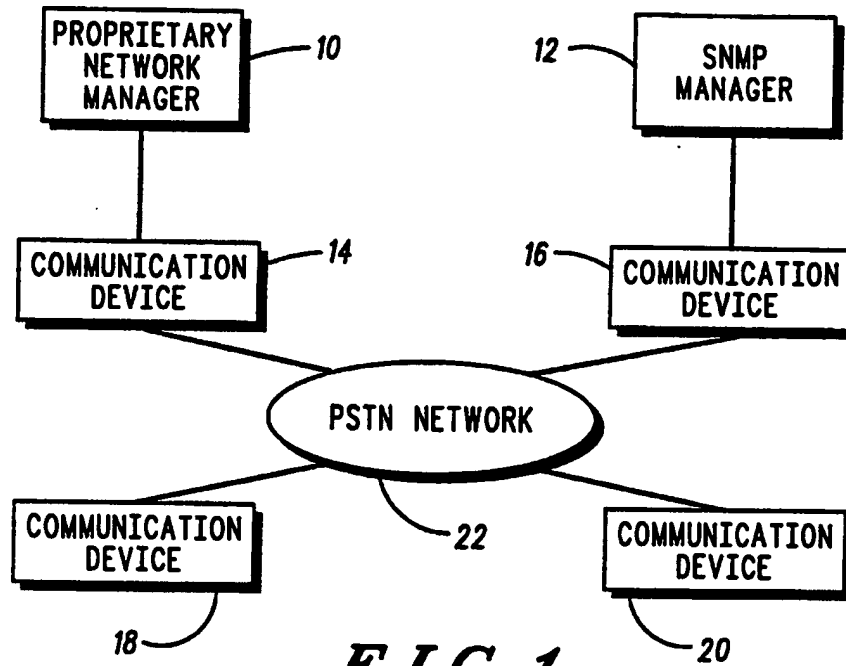
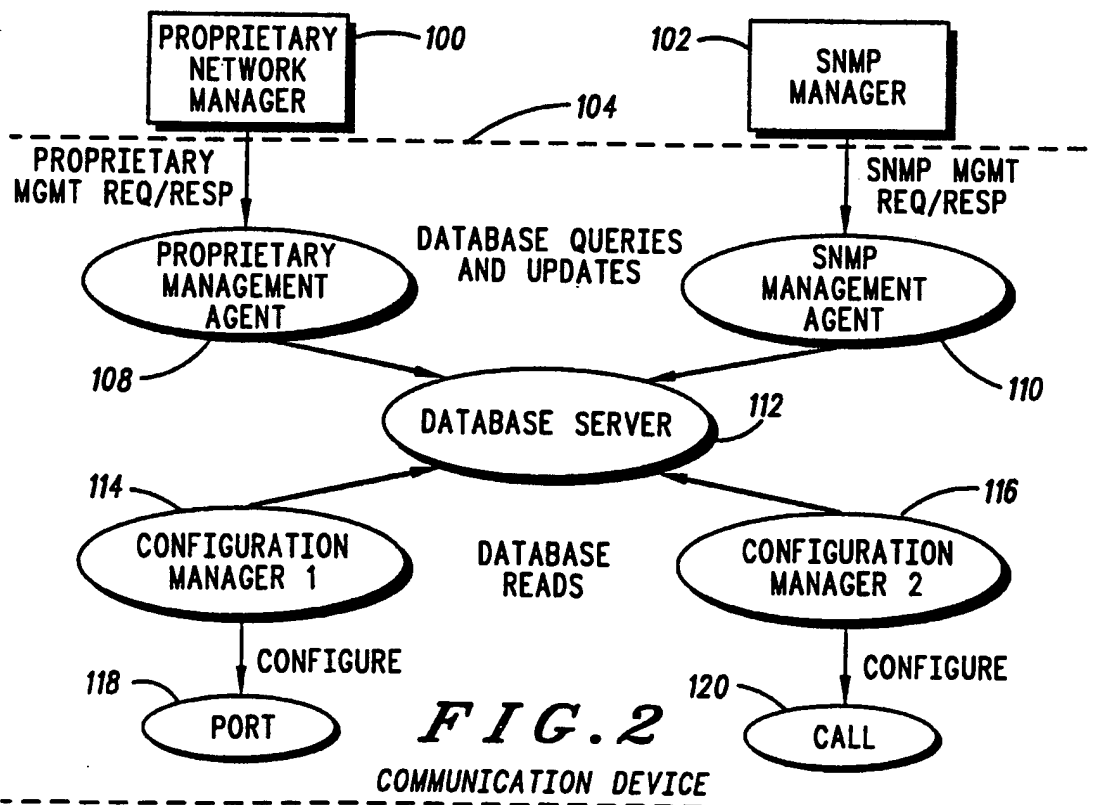
15 7. A method of configuring a communication device comprising the steps of:
 storing configuration information about a communication device in a communication device database;
 reading the configuration information from the communication device database;
20 configuring the communication device from the configuration information.

25 8. The method of claim 7 where the step of storing configuration information includes the step of converting the configuration information into third normal form and the step of reading the configuration information from the communication device database includes converting the configuration from third normal form into a configuration information organization recognizable by the communication
30 device.

35 9. The method of claim 8 where the step of configuring the communication device includes sending a command to the communication device instructing the communication device to begin configuration.

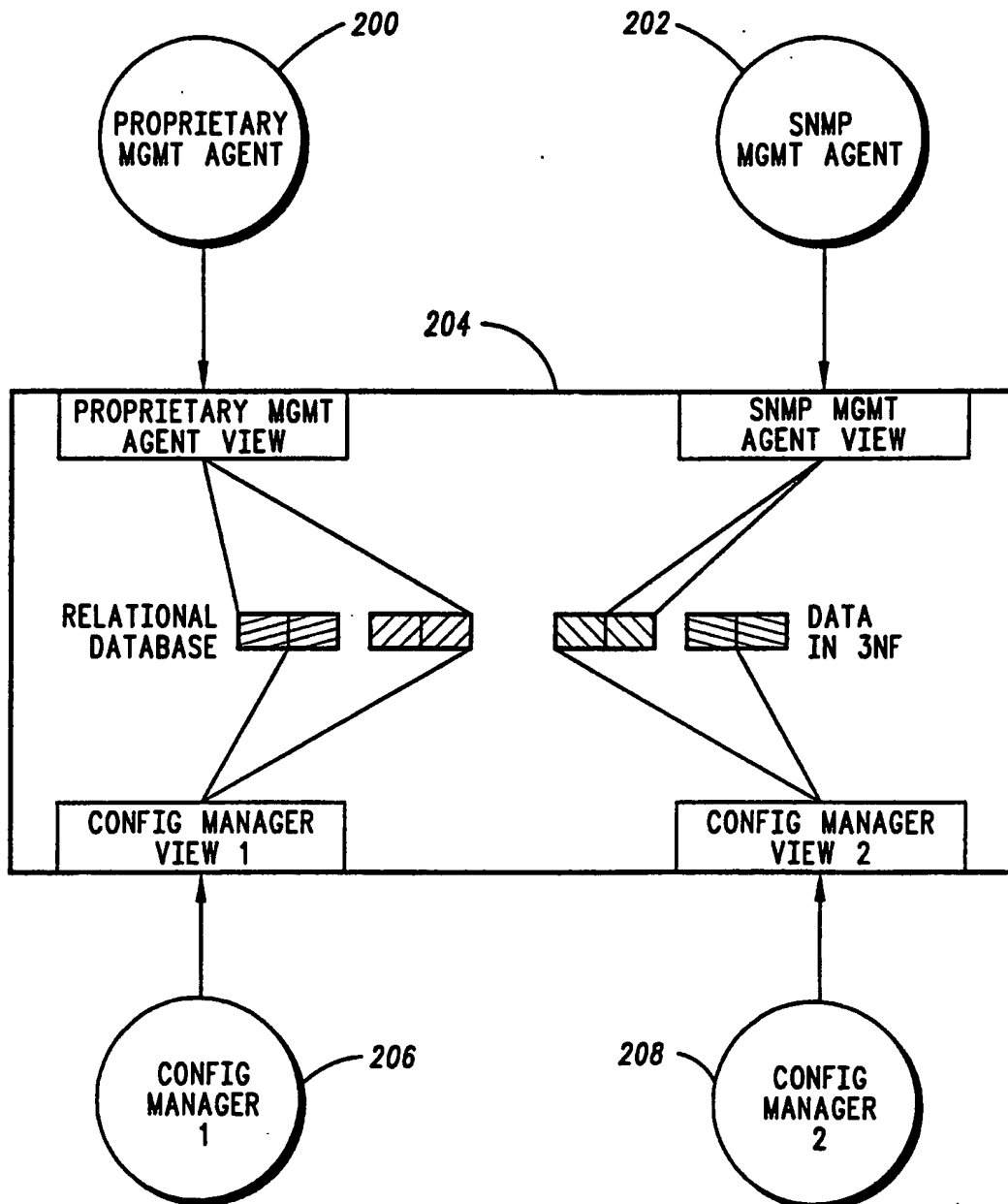
10. The method of claim 9 where the step of configuring the communication device includes the step of updating one of more of the versions of the configuration data in the
- 5 communication device database.

1/4

**FIG. 1****FIG. 2**

COMMUNICATION DEVICE

2 / 4

**FIG. 3**

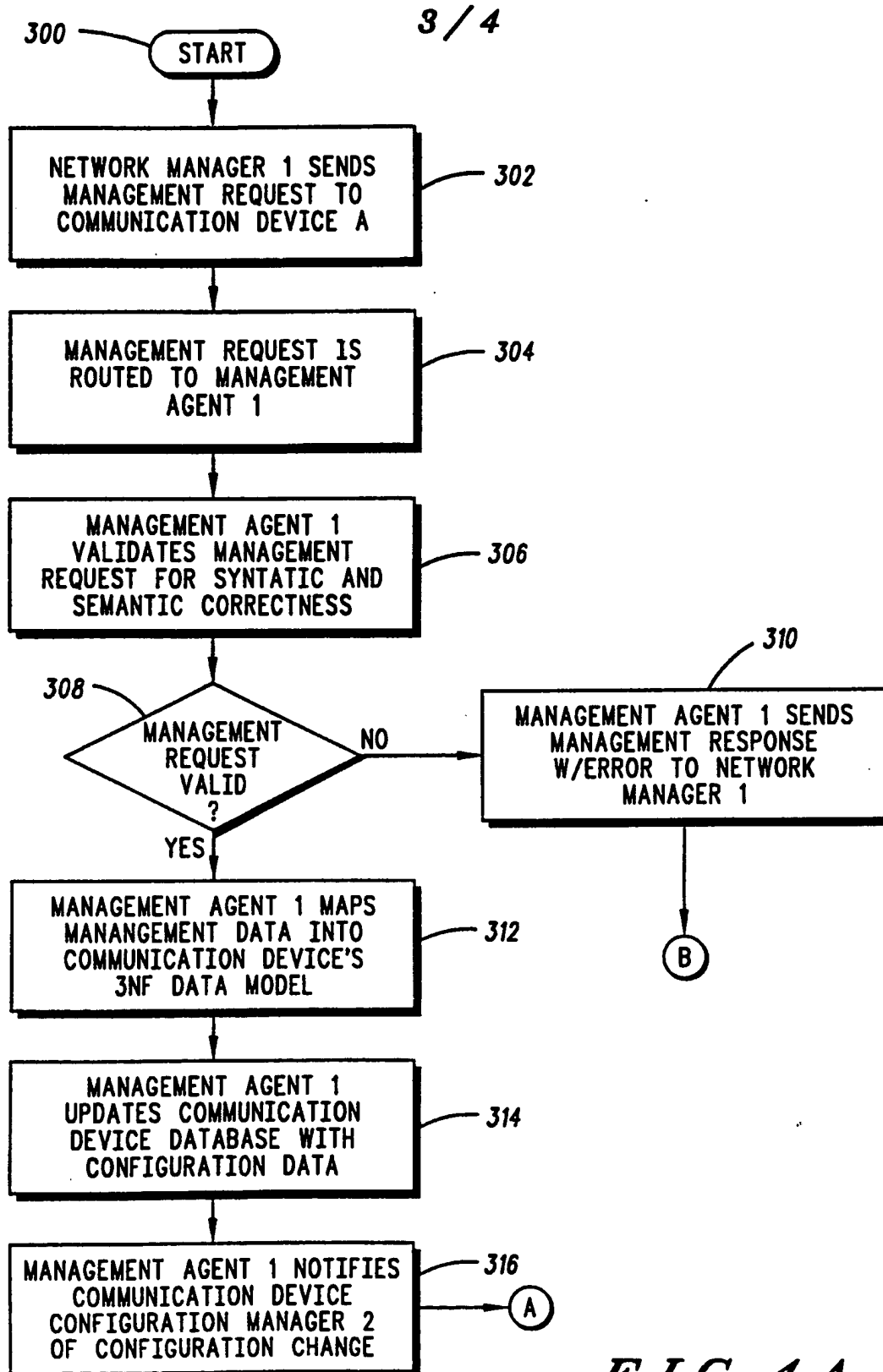
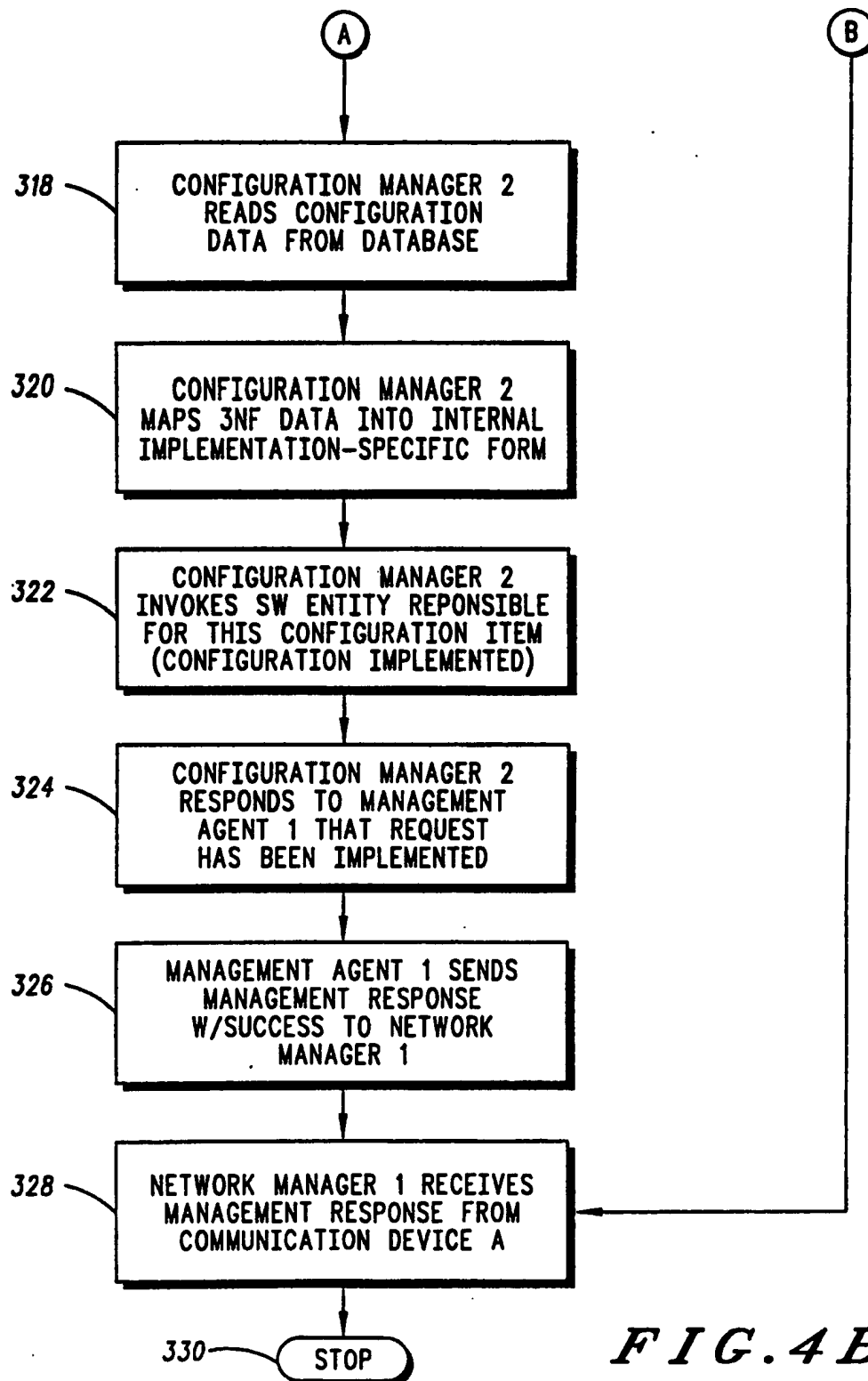


FIG. 4A

4 / 4

*FIG. 4B*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US94/08934

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 3/00, 13/00
US CL :395/200, 500, 600,800

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/200, 500, 600, 800

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS US FILE, IEEE INSPECT FILE, DIALOG DATABASE

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X — Y	US, A, 5,136,716 (Harvey et al) 04 August 1992, see figs. 1 and 3A; col. 2, lines 26-50, col. 4, lines 33-35	1-2, 7 ----- 3-6, 8-10
Y	US, A, 5,165,018 (Simor) 17 November 1992, see fig. 1; col. 4, lines 35-66; col. 48, lines 5-6.	1-10
Y	US, A, 4,622,633 (Ceccon et al) 11 November 1986, see col. 1, lines 9-11; col. 3, lines 36-38; col. 47, lines 39-62; col. 48, lines 44-47.	1-10

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* *A* *E* *L* *O* *P*	Special categories of cited documents: document defining the general state of the art which is not considered to be part of particular relevance earlier document published on or after the international filing date document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified) document referring to an oral disclosure, use, exhibition or other means document published prior to the international filing date but later than the priority date claimed	*T* *X* *Y* *Z*	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art document member of the same patent family
--------------------------------------	--	--------------------------	--

Date of the actual completion of the international search

08 DECEMBER 1994

Date of mailing of the international search report

JAN 20 1995

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231

Facsimile No. (703) 305-3230

Authorized officer

MENG-AI T. AN

Telephone No. (703) 305-9678

PCTWORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁶ : G06F 17/30	A1	(11) International Publication Number: WO 98/40829 (43) International Publication Date: 17 September 1998 (17.09.98)																														
<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p>(21) International Application Number: PCT/AU98/00162</p> <p>(22) International Filing Date: 12 March 1998 (12.03.98)</p> <p>(30) Priority Data:</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;">PO 5601</td> <td style="width: 40%;">12 March 1997 (12.03.97)</td> <td style="width: 30%;">AU</td> </tr> <tr> <td>PO 5638</td> <td>13 March 1997 (13.03.97)</td> <td>AU</td> </tr> <tr> <td>PO 8636</td> <td>15 August 1997 (15.08.97)</td> <td>AU</td> </tr> </table> <p>(71) Applicant (for all designated States except US): CLEVER-WORX INC. [US/AU]; Suite 168, 45 Cribb Street, Milton, QLD 4109 (AU).</p> <p>(72) Inventor; and (75) Inventor/Applicant (for US only): OLIVER, Brian, Keith [AU/AU]; 18/35 Troughton Road, Robertson, QLD 4109 (AU).</p> <p>(74) Agent: PIZZEYS PATENT & TRADE MARK ATTORNEYS; Level 6, 444 Queen Street, Brisbane, QLD 4000 (AU).</p> </div> <div style="width: 48%;"> <p>(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).</p> <p>Published <i>With international search report.</i></p> </div> </div>			PO 5601	12 March 1997 (12.03.97)	AU	PO 5638	13 March 1997 (13.03.97)	AU	PO 8636	15 August 1997 (15.08.97)	AU																					
PO 5601	12 March 1997 (12.03.97)	AU																														
PO 5638	13 March 1997 (13.03.97)	AU																														
PO 8636	15 August 1997 (15.08.97)	AU																														
<p>(54) Title: A COMPUTERISED METHOD FOR DYNAMICALLY CREATING, MODIFYING, REMOVING AND MAINTAINING INFORMATION IN A DATABASE</p> <p>(57) Abstract</p> <p>A method of retrievably storing in a computer database, aggregates each consisting of related titled information, the method including: storing each title as a conventional record in a conventional database so that titles can be efficiently added, moved or removed as conventional records; providing further database means in which individually titled information of an aggregate may be stored as a conventional record in a further conventional database; and operatively associating each individually titled information with its title and aggregate whereby each aggregate may be stored, removed, updated or queried.</p>																																
<div style="display: flex; justify-content: space-between; align-items: flex-start;"> <div style="width: 48%;"> </div> <div style="width: 48%;"> <p>Numeric Cell Repository Table</p> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <tr><td>1</td><td>1</td><td>5123</td></tr> <tr><td>1</td><td>2</td><td>18</td></tr> <tr><td>2</td><td>1</td><td>6471</td></tr> <tr><td>2</td><td>2</td><td>26</td></tr> <tr><td>3</td><td>1</td><td>9880</td></tr> <tr><td>3</td><td>2</td><td>28</td></tr> <tr><td>4</td><td>1</td><td>3362</td></tr> <tr><td>4</td><td>2</td><td>22</td></tr> </table> <p>Numeric Column Definition Table</p> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <tr><td>1</td><td>Employee#</td><td>9999</td></tr> <tr><td>2</td><td>Age</td><td>150</td></tr> </table> </div> </div>			1	1	5123	1	2	18	2	1	6471	2	2	26	3	1	9880	3	2	28	4	1	3362	4	2	22	1	Employee#	9999	2	Age	150
1	1	5123																														
1	2	18																														
2	1	6471																														
2	2	26																														
3	1	9880																														
3	2	28																														
4	1	3362																														
4	2	22																														
1	Employee#	9999																														
2	Age	150																														

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakhstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

**"A COMPUTERISED METHOD FOR DYNAMICALLY CREATING,
MODIFYING, REMOVING AND MAINTAINING INFORMATION IN A
DATABASE"**

5 This invention relates to a computerised method for
dynamically creating, modifying, removing and maintaining
information in a database and databases so formed.

 In particular this invention is concerned with a method
of organizing, storing, representing, retrieving, indexing
10 and querying uniformly structured information in a
conventional database. However it is to be understood that
this invention may also be utilised for storing and
manipulating non-uniformly structured information.

BACKGROUND OF THE INVENTION

15 Much of what people and organizations do in the work place is
intimately tied up with the use, reuse and expansion of
existing knowledge in reports, assessments and other
documents, expertise, facts strategies and other information.
Computer technology has enabled users to capture, manage,
20 structure, maintain and deploy that knowledge, however the
current systems are often costly, expert dependent,
inefficient, labour intensive, and time consuming.

 In order that existing art and this invention may be
readily understood and put into practical effect reference
25 will be made hereinafter to the accompanying drawings
wherein:-

FIG. 1 is an outline of a computer;

FIG. 2 provides Employee Information to be represented
in a database;

30 FIG. 3 provides the conventional representation of
Employee Information from Fig. 2 in a conventional
database table;

FIG. 4 illustrates aggregates of information;

FIG. 5 represents Employee Information from Fig. 2 in a

conventional database table but according to this invention;

FIG. 6 illustrates adding a new column of information to a database (Fig. 3) where information is organised in a conventional manner;

FIG. 7 illustrates an algorithm to add a new column to a database containing information organised according to this invention;

Fig. 8 illustrates adding a new column of information to the database as outlined in Fig. 5;

FIG. 9 is an algorithm to delete an existing column from a database containing information organised according to this invention;

FIG. 10 illustrates the Numeric Column Definition and Cell Repository Tables from Fig. 5 after the Removal of the Salary "Column";

FIG. 11 illustrates the renaming a column from Fig. 5;

FIG. 12 illustrates storage space for incomplete or undefined cell values are wasted in a record using a conventional method of representing information in a database;

FIG. 13 illustrates storage space for incomplete or undefined cell values which are not stored in a database organised according to this invention;

FIG. 14 illustrates an algorithm for adding a new row of information to a database containing information organised according to this invention;

FIG. 15 illustrates the result of Adding a row to a Database whereby information is represented according to this invention, as in Fig. 5;

FIG. 16 illustrates an algorithm for retrieving an existing row of information from a database containing information organised according to this invention;

FIG. 17 illustrates an algorithm to update an existing row of information in a database containing information

organised according to this invention;

FIG. 18 illustrates the result of updating Row #3 using the algorithm as detailing in Fig. 16 in a Database whereby information is represented according to this invention, as in Fig. 5;

FIG. 19 illustrates an algorithm for deleting a row from a database containing information organised according to this invention;

FIG. 20 illustrates the result of Deleting Row# 2 using the algorithm as detailed in Fig. 19 from a Database whereby information is represented according to this invention, as in Fig. 5;

FIG. 21 illustrates SQL Queries for both information represented in a database using a conventional manner and according to this invention;

FIG. 22 illustrates linking of the Employee Information of Fig. 2 in a conventional database table according to this invention using a "linked-list" arrangement;

FIG. 23 illustrates a method of simplifying the year 2000 changeover by regenerating only "year" cell values in a Cell Repository Table;

FIG. 24 illustrates adding new a type of column by adding a new Column Definition and Cell Repository Table, and

FIG. 25 illustrates placing individual columns of information as in Fig. 2 in separate tables in a database for separate column indexing.

DESCRIPTION OF A COMPUTER

For the purposes of this application the term computer includes a machine of the general type as outlined in Fig. 1. Typically a computer optionally includes a keyboard input device, a pointing device, a plurality of display devices, mass storage means, printing means, network communication means, an input/output controller, main memory and a

plurality of central processors possibly associated cache memory, connected by a communication mechanism or bus of some description.

CONVENTIONAL DATABASE SYSTEMS

5 The purpose of a conventional database system is to organize, store, process, index, query and report upon information. Usually prior to information being stored in a database, the values and types information to be stored undergo a formal engineering process called normalization,

10 (<revisit - reference to database systems books here>), although some instances of information need not be normalized as it may already be in a normal form.

 Generally one of the major reasons why normalization is performed is to organize the information to be stored in a
15 database into an efficient structure for storage and retrieval. More specifically during the process of normalization, information, commonly represented in terms of cell values, organized into columns, and rows as in Fig. 2, is organized into database tables so that cells values may be
20 easily, and efficiently stored, retrieved, queried and reported upon by conventional database systems.

 Generally, the level of information normalization is of great significance because the level of normalization not only influences the relative speed of the database operations
25 on the stored information, but also the efficiency of information storage.

 The historical importance of performing information normalization prior to storage in a database is further exemplified in its teaching in foundational database courses
30 at Universities and other similar institutions, the extensive coverage of normalization in database texts and database system documentation and the often claimed dependence on having normalized databases for efficient processing with

database query languages such as SQL (Structured Query Language).

Prior to normalization, information is often represented in terms of cells containing values, organised into rows and 5 columns, whereby columns represent similar types of cell values and rows represent related or associated cell values. For example, in Fig. 2, individual cell values are organised into columns according to the type of information, i.e. Names, Ages, Address, and each row is organised to represent 10 related cells values for a particular item, abstraction, object or thing. In the case of Fig. 2, each row represents the information cell values for a particular employee in an organization.

After normalization, information cell values, rows and 15 columns are often represented in terms of database record values, records, fields and tables. (Again, it is recognised that particular information columns and rows destined for storage in a database may not need to undergo normalization as the information may already be in an efficient normal 20 form).

In most situations once normalization is complete, there is often a one-to-one relationship between the representation of information as cells, columns and rows and how the information is stored in a conventional database system as 25 record field values, fields and entire records respectively. For example, Fig. 3 outlines how the information represented in Fig. 2 may be represented in a conventional database table.

DESCRIPTION OF A CONVENTIONAL DATABASE

30 In conventional computer database terms a database consists of tables, fields and records whereby;

a database consists of one or more tables;

a table consists of zero or more records;

the structure of the records in a table is defined by

one or more fields;

5 a field specifies the name, type and other necessary storage requirements that define the representation of a single piece of data. (eg. a name field as text with fifty characters, and age field as number, etc.). In most circumstances, a field is used to represent the definition of a column;

10 a record is a collection of field values, the structure of which is defined by the fields, the values being the actual information cell values in the database. In most circumstances, a record is used to represent a row of information, and

15 a field value is a single piece of information to store in a database. In most circumstances, a field value represents a single information cell.

Because there is often a one-to-one relationship between the representation of the information to be stored in a database and the representation of information in a computer database, the terminology used to describe the structure of information, i.e. cells, rows, and columns, is often used interchangeably with the definition terms of a computer database, i.e. field values, records and fields, respectively. For example, examine the relationships between the definition of information to be stored in a database as in Fig. 2 and the actual representation of the information in a database as in Fig. 3.

LIMITATIONS OF A CONVENTIONAL DATABASE

30 In conventional database tables, the number and type of fields which define the structure of a database table and thus the information which can be stored in a table, is fixed once a table has been created. That is, new database table fields may not usually be added, or existing database table fields reordered or removed from a conventional database table without the need to rebuild existing records, or by

writing software to do so.

For example, in order to add a new column of information, say Height, represented by a field to the conventional database Employee Table represented in Fig. 3, a completely new Employee database table would need to be created containing the fields from the original Employee table, along with any new fields, in the example, the Height field.

Each and every individual field value from every individual record from the original Employee database table would then have to be copied into the new Employee Table. This is usually a very inefficient and time consuming process, especially when there are a large number of existing records that need to be processed. Fig. 6 outlines the process.

A similar regeneration process is often required when fields must be deleted, renamed or moved within a conventional database table. Often however there is generally no limitation on the timeliness of creation, renaming and removal of tables of information from a conventional database.

In order to allow for the need to add additional columns of information so that the new information may be stored in rows in a database, while avoiding the need to regenerate records, one of the methods currently available is to allocate empty (redundant) columns of a variety of types at the time of initial creation of a database table. That is the method prescribes leaving space for the future information.

The problem with this method is that information cells in a row reserved for the future are left unused or empty, their storage space, although not needed is redundantly maintained. This can greatly increase the size of a database. Even so, once all of the redundant columns are used, the entire database table must be regenerated so that more

redundant columns may be added.

Although the limitation of being unable to simply and efficiently add, organize or remove fields in conventional databases has not been particularly limiting in the past,
5 (mainly due to the allocation of redundant fields), in an ever changing information environment, where the content and structure of information being collected, processed and reported upon is volatile, this limitation is increasingly reached early.

10 The subsequent cost of maintenance on the database systems is thus increased both for individuals and organizations using database systems, especially in situations whereby there are a large number of existing records or software must be maintained or created to perform
15 such maintenance.

OBJECTS OF THIS INVENTION

The present invention aims to alleviate at least one of the above mentioned disadvantages and to provide a reliable and efficient method whereby the process of creating,
20 organizing, renaming, or moving columns of information in a database may be performed efficiently.

In our provisional specifications Nos. PO 5601 and PO 5638 filed in Australia on 12 March, 1997 and 13 March, 1997 in respect of which priority is claimed we refer to
25 "conventional" and "alpha" databases. The alpha database was described as being a conventional database storing alpha field definitions and alpha field data. However the term "alpha" in this art may be interpreted to have a separate significance as opposed to being a distinguishing name only
30 and accordingly the description "alpha" is not used in this specification. The term "aggregate" is now used to indicate an "alpha record" and the term "related titled information" now refers to the "alpha fields" forming an "alpha record". An "alpha database" is now referred to as "aggregates of

related titled information".

SUMMARY OF THE INVENTION

With the foregoing in view, this invention in one aspect resides broadly in a method of retrievably storing in a computer database aggregates each consisting of related titled information, the method including:-

storing each title as a conventional record in a conventional database so that titles can be efficiently added, moved or removed as conventional records;

10 providing further database means in which individually titled information of an aggregate may be stored as a conventional record in a further conventional database, and operatively associating each individually titled information with its title and aggregate whereby each
15 aggregate may be stored, removed, updated or queried.

The aggregates each consisting of related titled information may be illustrated as in Fig. 4. According to this invention, information may be randomly added, moved or removed without the need to regenerate any existing records
20 in the database.

This invention may utilise systems such as Microsoft Access, Microsoft FoxPro, Borland Paradox, Borland Dbase, Oracle Database Systems and the like. However the databases used to implement this invention may include database systems
25 such as relational, hierarchical, federated or object-oriented databases. If desired the databases could be purpose built to store the above required information.

In a further aspect this invention resides broadly in a method of storing information in conventional databases, the
30 method including:-

storing each column definition as a conventional record in a conventional database so that column definitions can be efficiently added, moved or removed as conventional records;

providing further database means in which cell values of

a record are stored as separate records, and
associating each cell value with its column.

Preferably associating each cell value with its column
is achieved by additionally storing a reference to a column
5 definition along with each cell value. Alternatively
associating each cell value with its column may be achieved
through the use of conventional database foreign keys.

By using this method, addition, removal or
reorganization of column definitions does not destroy the
10 integrity of existing cell values. Furthermore such
operations may occur without the need to regenerate any
existing records in a database.

According to a preferred form of this further aspect of
the invention, organisation of information may involve:

15 . establishing a conventional database table, hereafter
referred to as a Cell Repository Table storing either a
single column of cell values, a relatively small number
of related cell values, and/or columns of cells values
of the same type;

20 . establishing an additional conventional database
table, hereafter referred to as a Column Definition
Table to store the defining characteristics of a column
of information including a column's name and/or other
associated information pertaining to the definition of a
25 column;

. establishing a means of associating individual cell
values stored in a Cell Repository Table with their
respective column definitions in the Column Definition
Table.

30 From the above it will be understood that rather than
storing semantically related cell values such as rows of
records forming a table as in Fig. 3 where each employee row
from Fig. 2 is stored as one conventional record in a
conventional table, this invention provides that individual
35 columns of cell values are stored in separate conventional

tables such as in Cell Repository Tables as in Fig. 5, to more easily permit the addition of new columns and the removal and reorganization of existing columns to the table.

Furthermore, the information pertaining to the
5 definition of each column and its associated properties are stored as separate records in a Column Definition Table for each specific type of column. (Refer to the Column Definition Tables in Fig. 5.3 and Fig. 5.4).

For example, rather than storing cell values in
10 conventional records as in the Employee Table as illustrated in Fig. 3, all numeric information cell values from the rows for the Employee Table, Fig. 3, may be stored in a single numeric Cell Repository Table as illustrated in Fig. 5.1, and all string information cell values from the rows for Employee
15 Table, see Fig. 3, may be stored in a separate string Cell Repository Table as illustrated in Fig. 5.2.

In addition, the information pertaining to the definition of each of the column names in the Employee Table may be stored as separate records in the appropriate Column
20 Definition Table, as in Figs. 5.3, and 5.4.

As all of the cells values of the same type are stored in a single Cell Repository Table, a means of differentiating the different types of cell values for the different types of columns stored within a Cell Repository Table is required so
25 that particular cell values for a specific column may be created, accessed, maintained or removed for a specific row of information.

In order to identify the column in which each cell value in a Cell Repository Table has been stored, it is preferred
30 that reference is additionally made to the column's definition stored in a secondary conventional database table, called a Column Definition Table. This may be achieved through uniquely identifying each column in a column definition table with a unique number and storing this number
35 with each cell value stored in a Cell Repository Table.

Additionally, the name of a column stored in a Column Definition Table along with any other parameters a column required for its definition, such as size, range and precision is also stored. See Fig. 5.3 and 5.4.

5 Furthermore, each cell value stored in a Cell Repository Table also stores the row number and column identifier that the cell belongs to.

As a result individual rows and specific cell values may be accessed, updated, queried, indexed and maintained just as
10 records may be organised in a conventional manner fields and records may be accessed, updated, queried, indexed and maintained.

In conventional database terminology, the row number and unique column identifier values of the records in the Cell
15 Repository Tables act as "foreign keys" so that access to a specified cell value for a specific row and column may be readily performed. Thus for every Cell Repository Table, there is at least one associated Column Definition Table and vice-versa.

20 The use of a Column Definition Table is exemplified by examining either the numeric Column Definition Table in Fig. 5.3, or the string Column Definition Table in Fig. 5.4. Both Column Definition Tables define unique column identification numbers for each different type of column in the Employee
25 information as represented in Fig. 2, along with the names of the columns and other appropriate column parameters. The Column unique column identification numbers are then used as foreign keys in the Cell Repository Tables, see Fig. 5.1 and Fig. 5.2, to identify which of the defined columns a
30 particular cell value pertains to.

For example, in Fig. 5.1, the record <2, 1, 6471> represents the information for row 2, numeric column 1 (which, by looking up the numeric column definition in the numeric Column Definition Table is the "Employee#" column) in
35 the employee information as represented by Fig. 2.

Additionally, in Fig. 5.2, the record <4, 2, Cairns> represents the information for row 4, string column 2 (which, by looking up the string column definition in the string Column Definition Table is the "City" column) in the employee information, again as represented by Fig. 2.

In yet a further aspect this invention resides in a database structure including:-

a conventional database table established as a Column Definition Table for storing the defining characteristics of a column including a column's name and other associated information pertaining to the definition of a column;

a further conventional database table established as a Cell Repository Table storing a column of cell values and a column of associated row values, and

associating means for association individual cell values stored in a Cell Repository Table with their respective row and column values from the Column Definition Table.

20 Description of a typical embodiment

A typical embodiment of this invention resides in the organisation of information in a database system called CleverX, detailed information of which is illustrated in the user manual attached.

CleverX is a flexible and sophisticated information and assessment capture tool which uses a conventional database system to store and organize information according to one aspect of this invention and whereby in CleverX terminology;

a CleverX Workbook is a database containing information organised according to this invention;

other CleverX palette components, such as CleverX Worksheets, CleverX Sections, CleverX Responses and CleverX Textboxes are used to represent different types of columns of according to this invention.

As this invention efficiently permits the addition, modification and removal of columns of information within a database, CleverX similarly allows for the addition, modification and removal of CleverX Workbooks, CleverX
5 Worksheets, CleverX Sections, CleverX Responses and CleverX Textboxes, each of which is responsible for the collection and organisation of CleverX user determined information in CleverX.

Furthermore, this invention enables the storage of
10 additional properties along with each column definition, CleverX uses this feature to store additional CleverX Component definition properties such as user-interface appearance, security, user-defined properties and relationships to other CleverX components.

15 Additionally, CleverX makes use of the ability for information represented according to this invention to be incomplete or undefined, thus allowing a CleverX user to partially specify or leave CleverX component values incomplete. Furthermore, CleverX permits the definition of
20 conditions which allow a CleverX user to define when user information does not need to be recorded in a database. Under these circumstances, information provided by a CleverX user may be stored in as a non-uniform row in a database.

Adding a New Column of Information

25 As indicated above one of the limiting properties of conventional database systems is that most do not permit the addition of new fields to existing tables without the need to create a new table containing the new field(s), copy all of the existing records from the existing table into the new
30 table, and potentially re-indexing the new table. See Fig. 6.

Although this may not necessarily be a problem for

conventional database tables which contain a small number of existing records, or for database tables representing information that rarely changes, it does represent a significant problem for database tables with a large number of records and those that need to be changed often, as the process of regenerating an existing table can be costly in terms of development and processing time, the latter usually being proportional to the number of records in the database.

Although some of the newer database systems now permit the addition of new columns of information to database tables without resorting to the obvious construction of new tables or copying of existing records, they often consume an amount of time proportional to the number of existing records and/or do not permit the renaming, moving or removal of existing database fields.

By organizing the storage of information according to this invention, the addition of new columns of information into a database table may occur in a relatively short time, regardless of the number of existing records. Furthermore there is no need to regenerate or re-index existing records. This is because adding a new column simply requires the addition of a new record in a Column Definition Table. The algorithm for adding a new column is outlined in Fig. 7.

For example, in order to add a new column to store an employee's height, in a conventional database system the process as in Fig. 6 is often used. Whereas by organizing information according to this invention, addition of the height field would be as in Fig. 8. A consequence of not storing related cell values in a row is that there is no need to reserve storage space in each record for potential new columns as the new cell values for new columns will be stored separately in a new or existing table.

Removing an Existing of Column of Information

A further limiting characteristic of conventional

database tables is that removal of existing columns of information also requires regeneration of table records, just as required when adding new columns of information to a table. However, by organizing the storage of information using the method of this invention, the removal of a column of information only requires:

the removal of the specific column from it's Column Definition Table, and

the removal of the specific column's associated cells as stored in the associated Cell Repository Tables.

The algorithm which may be used for this purpose is outlined in Fig. 9.

The reason why record regeneration its not required is because, removing a column only requires removal of a column definition record from a Column Definition Table, and removal of the column cell values from the associated Cell Repository Table. Regeneration is not required because individual cell values are stored in separate database tables.

For example, in order to remove the Salary column of information from the conventional representation of an Employee table in Fig. 3, a new Employee table would need to be created without the Salary column, and then all of the existing record values from the existing Employee table would need to be copied into the new table, minus the Salary column values.

Removal of the Salary column from a database according to this invention, requires only:-

removal of the Salary column definition from the numeric Column Definition Table illustrated in Fig. 5.3. This results in the table illustrated in Fig. 10.1, and then

removal of the Salary cell values from the numeric Cell Repository Table, see Fig. 5.1, with a conventional database structured query language statement such as the following, which results in the table illustrated in

Figure 10.2.

"DELETE RECORDS FROM numeric Cell Repository Table
WHERE NumericColumnID = 3".

5 Renaming and Moving Existing Column Definitions

Renaming or moving existing column definitions when representing information in a database using the conventional manner are not often performed due to the requirement that such operations usually require existing table records to be
10 the regenerated as in adding or removing field definitions.

Organizing database information according to this invention may be simply achieved by making changes to the appropriate column definition(s) in the Column Definition Table. Consequently there is no need to access the cell
15 values stored in the Cell Repository Table and thus regenerate existing records.

For example, in order to change the name of the "Age" column in the employee information as in Fig. 2 to "Years Since Birth", a database organized according to this
20 invention would only require the appropriate column in the numeric Column Definition Table to be changed, as illustrated in Fig. 11.

Applicability of Structural Changes

Another feature of information represented according to
25 this invention is that the aforementioned operations of creating new columns of information, or editing or changing a definition of a column within a database table may be performed randomly. Furthermore these operations may occur while new or existing cell values are being stored or updated
30 in a database. This may occur as storage of cell values in Cell Repository Tables occurs independently of the storage of column definitions in a Column Definition Table. Furthermore, as a column's name and other associated properties are independent of the storage of cell values,

column names and associated properties may also be changed simultaneously during operation of information in the database.

This offers a significant advantage over the conventional approach of organizing information in a database, because in most circumstances, modifying the structure, definition or organization of columns in a database requires that no other operations are simultaneously taking place. In many circumstances, databases storing information organized according to this invention do not need have networked users locked out during a structural update of a database.

Record Completeness and Non-Uniform Row Information

Another feature of representing information according to this invention is that storage space for cell values that are undefined, incomplete or unknown are not redundantly stored as in database systems that store information in a conventional manner.

For example, in Fig. 12, where the employee table has a Partner column included, storage space for those employees that do not have a partner is wasted, as a conventional record in a database reserves enough storage space for all of its field values regardless of whether all of the field values are actually used.

However, by organizing the information according to this invention, values of cells that are undefined, incomplete or unknown for a row are not stored. For example in Fig. 13 the partner cells are not stored for Mary and Dave. A consequence of this is that databases storing information according to this invention do not waste space when cell values are not used.

This feature allows row information to have non-uniform structure unlike rows stored in the conventional manner.

This consequently allows the representation of non-uniform row information, [[OUT(as described in US Patent #5682524),]] without the need to develop additional database software query mechanisms, without the need to represent row
5 information in a hybrid or unconventional binary format using single fields, likes BLOB (binary large objects) that conventional database query languages, like SQL, can operate upon, and without restricting the database tools that may access information organized according to this invention, as
10 most standard database development tools can not directly inspect BLOB fields without the development of additional software.

Adding a New Row of Information to a Database

As individual rows of information are not stored as
15 single records in a database table, the method by which entire rows of information are traditionally added to databases having their information organized in a conventional manner can not be used. Instead each individual cell value is added as a separate record to the appropriate
20 Cell Repository Table. This is illustrated in algorithm outlined in Fig. 14. Fig. 15 exemplifies the use of the algorithm outlined in Fig. 14 by showing the result of adding the new Row# 5 containing the cell values represented as the set {(Employee#, 1234), (Name, Kevin), (Age, 31), (City,
25 Melbourne), (Salary, 52000)} using the algorithm as detailed in Fig. 14.

Retrieving an Existing Row of Information from a Database

Similarly, as individual rows of information are not stored as single records in a database table, the method by
30 which entire rows of information are traditionally retrieved from databases having their information organized in a conventional manner can not be used. Instead each individual cell value is retrieved from the appropriate Cell Repository

Table. This is illustrated in algorithm outlined in Fig. 16. As an example of the algorithm outlined in Fig. 16, the result of requesting row "2" from the information as stored in Fig. 5, would result in the set {(Name, Mary), (Age, 26),
5 (City, Brisbane), (Salary, 42000)} being returned.

Updating an Existing Row of Information in a Database

Similarly, as individual rows of information are not stored as single records in a database table, the method by
10 which entire rows of information are traditionally updated in a database having their information organized in a conventional manner can not be used. Instead each individual cell value is updated in its record stored in the appropriate Cell Repository Table. This is illustrated in algorithm
15 outlined in Fig. 17. As an example of the algorithm outlined in Fig. 17, Fig. 18 highlights the result of updating row "3" in Fig. 5 with the cell values {(Salary, 41000)}.

Unlike updating a row of information stored in a database in a conventional manner, when updating a row in a
20 database according to this invention, not all of the cells in a row need be updated.

Deleting an Existing Row of Information from a Database

When deleting, each individual cell value is deleted from the appropriate Cell Repository Table. This is
25 illustrated in algorithm outlined in Fig. 19. Fig. 20 exemplifies the use of the algorithm outlined Fig. 19 by showing the result of deleting row "2" from the database as outlined in Fig. 5.

Performing a Query on Information stored in a Database

30 An advantage of this invention is that it uses conventional database technology to store information. This enables operations that are applicable to information stored

in a conventional manner to be applicable to information stored in a manner according to this invention.

Consequently, structure query languages such as SQL, indexing systems and reporting tools may be used on database
5 storing information according to this invention. An illustration of this ability is outlined in Fig. 21 which highlights the use of conventional SQL queries on information both represented in the conventional manner and according to this invention.

10 Indexing Information in a Database

The generation and maintenance of indexes of cell values stored in a conventional manner are also applicable to cell values stored according to this invention. For example, an index created on the Name field in the Employee Table (Fig.
15 3), could be created on the string value field in the string Cell Repository Table in Fig. 5. Importantly however, the creation such an index over the string values in the Cell Repository Table in Fig. 5, would have the additional side-effect of indexing non-Name column values as the other string
20 cell values are also stored in the string Cell Repository Table. That is, indexing the value field in the string Cell Repository Table in Fig. 5 not only indexes Name column values, but also City column values as well. Although in some circumstances this additional indexing may not be
25 necessary, this can be overcome by creating an additional Name Cell Repository Table to store only the name string information values. This is illustrated in Fig. 25 for example.

Preferred Methods to Organize Information

30 While this invention has been devised to provide efficient management of columns of information, a result is that the retrieval of entire rows of cell values is not as efficient as storing rows of cell values as records in a

conventional database. However this may be alleviated by adopting a conventional linked-list approach to storing cell values whereby each cell value additionally stores associative information to link it to the next cell value in a row. Fig. 22 illustrates use of this method to represent the information presented in Fig. 5. However a characteristic of a linked representation is that the deletion of columns of cell values may require the regeneration of existing links between cell values.

Suitable algorithms for the addition and update of columns may be as previously mentioned. Suitable algorithms for the addition, update, retrieval and deletion of entire rows of information are standard linked-list algorithms.

Record Locking

Another advantage provided by this invention is that in a multi-user database environment, individual cell values may be locked during update. The benefit of this is that one or more database users may thus simultaneously update different cell values in the same row.

For example, if one user in an organisation wishes to update the cell value that represents the Name column of row #1 in the Employee Table of Fig. 3, while another user, potentially in another part of the organisation, simultaneously wishes to update the cell value that represents the Salary column also in row #1 in the Employee Table (Fig. 3), both updates may occur at the same time.

This is contrary to the conventional organisation of information in most database systems where only one user has the ability to update a particular record at any one time.

Regenerating of Cell Values in a Database

Another aspect of this invention involves the ability for column definition properties to be changed and their corresponding cell values in Cell Repository Tables to be

regenerated without affecting the integrity of other row cell values. This has the benefit that as the storage requirements for columns of cell values change over time, the column definition may be updated and its corresponding cell values regenerated without affecting other row cell values.

An example of this characteristic becomes apparent when considering transition to the year 2000. By organising information according to this invention, only the year cell values in a Cell Repository Table would need to be regenerated, rather than all of the conventional database table records containing 2-digit year values as in the current crisis situation. (See Fig. 23)

Adding new Types of Columns

Another advantage of this invention is that just as new columns of information may be added so to may new types of columns may be added.

For example, in addition to the ability to add a new column of an existing type to a database representing information according to this invention, See Fig. 7, it is also possible to add new types of Column Definition and Cell Repository Tables to the database so that new types of columns may be added to the information rows.

This feature is illustrated in Fig. 24, whereby both a date Column Definition and date Cell Repository Table have been added, allowing date cell values to be stored for employee information rows.

Benefits of this Invention

This invention is particularly useful in industries where the content and structure of information to be collected, stored, compiled, processed and reported upon is not completely known when a database is to be established. For example, any industry that performs assessments, surveys, makes extensive use of notes, require judgments to be made,

quotations, estimates and alike, or where the structure and content of information is volatile, could make extensive use of systems implementing this invention.

For example individuals and organizations such as
5 teachers, doctors, lawyers, banks, insurance and marketing companies, and alike where the information and knowledge they collect may change in structure and organization and may significantly do so over time as their information may be tailored efficiently with minimal database system disruption.

10 It will of course be realised that the above has been given only by way of illustrative example of the invention and that all such modifications and variations thereto as would be apparent to persons skilled in the art are deemed to fall within the broad scope and ambit of the invention as is
15 defined in the appended claims.

CHAPTER ONE

INTRODUCTION

Welcome to CleverX

CleverX is a revolutionary new tool for the management of expert knowledge.

CleverX substantially reduces the time and effort involved in producing reports about expert assessments of an object or situation. Almost any task that makes repeated use of expert knowledge can be streamlined by using CleverX.

Users of CleverX include:

- School teachers and university academics who spend many hours marking and providing feedback on large numbers of student assignments.
- Doctors and other health professionals who produce assessments or reports about patients that they deal with.
- Engineers, quality managers and auditors who spend large amounts of time conducting inspections, audits or assessments and writing reports about each audit.
- Employees in all sorts of businesses who spend many hours preparing similar or repetitive quotations, proposals, and tender responses.
- Marketing researchers and pollsters who spend large budgets conducting surveys and collating the results.
- Employees in all sorts of businesses who spend lots of time preparing orders.
- Police and military personnel who spend much time writing reports about cases, events or situations they are involved with.

In all of these situations, and countless others, CleverX can be used to make very substantial quality and productivity gains.

Where to Find Things in this Manual

This manual provides instructions on how to work with the CleverX application. It assumes that you are familiar with the Microsoft Windows environment and that you can use applications such as the Windows Explorer.

The remainder of this chapter contains information on how to install and remove CleverX, as well as some other house keeping information such as how to contact CleverworX.

The remainder of this manual covers the following topics:

CHAPTER 2 – Understanding CleverX

Chapter 2 contains an overview of the way CleverX works and how you can benefit from using CleverX. You should read this Chapter to gain a high level understanding of the main concepts associated with CleverX.

CHAPTER 3 – Your First CleverX Project

Chapter 3 is a step-by-step tutorial on creating and using a CleverX Project. If you want to get started immediately after installing CleverX then you can go straight to Chapter 3, but you may want to return to Chapter 2 after you have finished this tutorial.

Installing CleverX

You install CleverX onto your computer using a standard setup program. This setup program is provided as part of the CleverX package and is called `setup.exe`.

The setup program installs CleverX, this User's Guide, and some sample CleverX projects into the directory that you specify.

Release Notes

The README file that is part of the CleverX package contains information you need to know about the current release of CleverX. You should read this file before installing CleverX.

Hardware and Software Requirements

To run CleverX you must have at least the following hardware and software:

- An IBM compatible PC with a 486 processor or higher
- Microsoft Windows 95 or Microsoft Windows NT, version 3.51 or higher
- At least sixteen megabytes of memory
- At least 10 megabytes of free hard disk space

- A mouse, keyboard and VGA screen (or better)

Installing the CleverX Internet Package

CleverX is distributed from the CleverX web site.

To Download CleverX from the CleverX Web Site

- 1 Using your Web browser, go to the CleverX Web site at <http://www.cleverx.com>.
- 2 Go to the download page.
- 3 Click on the link for the CleverX distribution.
- 4 Save the CleverX package to a directory on your hard drive.

Note: Once you have downloaded the CleverX package, you must run the setup program to install CleverX. You cannot run CleverX directly from the Internet download.

To Install CleverX from the Internet Package

Note: Before installing CleverX, you need to remove any previous versions of CleverX that may be installed on your machine. See *Removing CleverX*, below.

- 1 Close any applications, other than the Windows Explorer, that may be running on your machine.
- 2 Using the Windows Explorer, go to the directory on your hard drive that contains the CleverX package.
- 3 Run `setup.exe`
- 4 Follow the instructions on your screen

Removing CleverX

To Remove CleverX From Your Computer

- 1 Make sure that CleverX is not running
- 2 From the Windows Start menu select **Settings** then **Control Panel**
- 3 In the Control Panel double click **Add/Remove Programs**
- 4 Go to the **Install/Uninstall** tab.
- 5 Select CleverX from the list of programs.
- 6 Click the **Add/Remove** button
- 7 Follow the instructions on your screen

Document Conventions

The following typographic conventions are used in this manual:

Example	Description
Delete	Words in bold are used to indicate the names of CleverX buttons or menu items.
File Save	A vertical bar is used to indicate items in a menu hierarchy. For example the Save menu item on the File menu.
type this	Words that you type appear in courier font.
"component name"	Names you have given to CleverX components appear in quotes in times font.

Contacting CleverworX

CleverX is produced by CleverworX Inc., Suite 670, 111 North Market Street, San Jose, California, 95113-1101. You can contact CleverworX by phone on (USA) 408-3676125.

The CleverX web site is at <http://www.cleverx.com>.

E-mail orders can be sent to sales@cleverx.com.

Support Requests

Support requests can be sent by e-mail to support@cleverx.com or submitted via our web page. We are committed to ensuring that CleverX is a high quality product. You can assist us to fix any problems that you may have with CleverX by providing the following information with any support request:

- The CleverX version number (available from the Help | About screen)
- The operating system you are using (Windows 95 or NT) and the version number of the operating system.
- The exact wording of any messages that CleverX displayed.
- A description of what you were doing when the problem occurred
- A step by step description of what we can do to reproduce the problem

- Any CleverX project files (.cx) that you may have that will allow us to reproduce the problem.
- The CleverX log file, which you can find in \Windows\Temp\cx.log on Windows 95 or \WinNT\Temp\cx.log on the NT operating system.
- Any other software (with versions) you have installed on your computer

CHAPTER TWO

UNDERSTANDING CLEVERX

The purpose of this chapter is to explain a number of concepts about CleverX that will help you to get the best results from the software, and will make the rest of this manual easier to understand. If you would prefer to start using CleverX immediately, and come back to this chapter later, then you can skip to Chapter 3, which presents a tutorial introduction to CleverX.

What CleverX Does

CleverX delivers significant quality and productivity gains in knowledge management and knowledge deployment. CleverX can be used to capture, structure, deploy, and maintain expert knowledge in numerous fields including education, health, medicine and business.

Examples of expert knowledge includes the feedback that a teacher would give to school or university students about their assignments; information a doctor would write in a report about a patient's condition; or details associated with quoting a price for a job or for an order.

In all these areas, and many more, CleverX offers substantial advantages by reducing the time and effort involved in performing tasks requiring expert knowledge.

Efficient Capture of Expert Knowledge

CleverX provides a simple yet powerful mechanism for capturing expert knowledge. Until now, the major inhibitors to the management of expert knowledge has been the time and cost associated with capturing, structuring, analyzing and maintaining that knowledge. CleverX not only captures expert knowledge but also substantially reduces the time of these daily tasks. These savings provide a powerful motivation for experts to use CleverX.

Powerful Structuring of Expert Knowledge

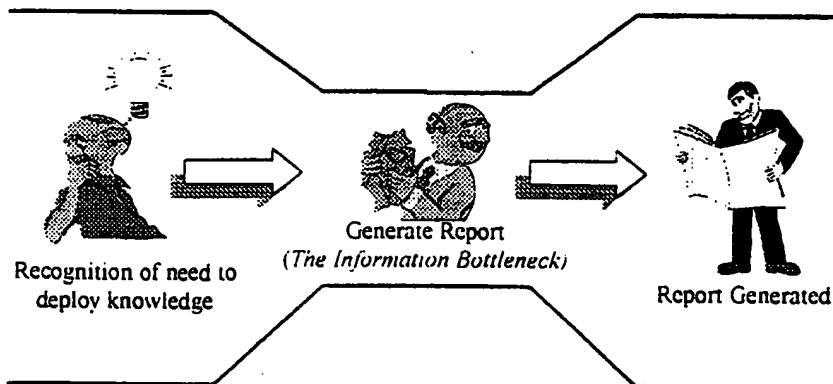
CleverX allows expert knowledge to be structured in a *knowledge tree*. Hierarchical structures allow easy access to and use of the knowledge, as it is required. At the highest level only general information is presented. Users can "drill down" into the tree as required. This structure amplifies the power of the knowledge and data stored by CleverX.

Efficient Deployment of Expert Knowledge

CleverX allows users to make use of expert knowledge much more efficiently than many other systems.

There are numerous situations where we quickly recognise that certain knowledge needs to be deployed but we end up using a very inefficient means to carry out the deployment. This is the *Information Bottleneck*. CleverX removes this bottleneck.

One example of the Information Bottleneck occurs when a teacher is marking an essay or an assignment. The teacher can very quickly recognise that a student's work has a particular problem with it, but the teacher will need to spend considerable time inefficiently writing down the same feedback so the student will be able to improve next time.



Because often, many students make the same mistake, this feedback may have to be laboriously written down over and over again.

The Information Bottleneck occurs because although the teacher has recognised the problem quickly, it will take time to document the problem in a report by writing down the details.

CleverX removes the Information Bottleneck by dramatically speeding up the deployment of knowledge.

Ongoing Maintenance of Expert Knowledge

One of the major disadvantages of most database and expert systems is that the structure of the information they contain is static and difficult to change. These systems almost always require a user to have some programming skills. This is not the case with CleverX. The structure of a CleverX Project can be very easily changed, even after you have finished using the Project to gather data.

This has several important implications:

- the expertise captured and used in a given CleverX Project *evolves and improves* over time
- *other people* can take advantage of and apply this expertise and thereby improve their own skills and performance, and
- there can be much greater *uniformity and consistency* in applying available expertise in a particular application area.

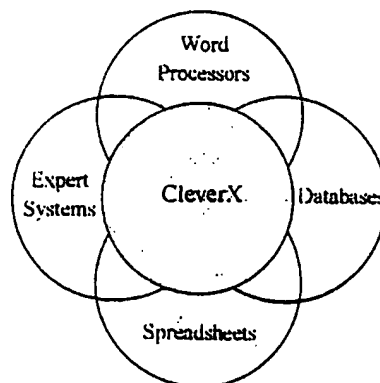
Standardised Information Used By Many People

By capturing and structuring knowledge in a CleverX Project, it is possible for many different people to apply this knowledge in a similar way. For example, if a CleverX Project contains criteria for marking student essays, then it is possible to distribute the project to many different teachers who can then apply the same criteria when marking.

No Programming Required

One of the major advantages of CleverX is that all its benefits can be realised without the need for you to resort to databases, expert systems, computer code or conventional information systems. Instead, CleverX provides a user friendly graphical user interface that is as simple to use as standard applications such as word processors or spreadsheets.

CleverX is not a database, nor an expert system or a word processor, or a spreadsheet, yet at the same time it may be used to solve, much more easily, problems that are often handled by such tools. Imagine there are four overlapping circles of application: one for *databases*, another for *expert systems* a third for *spreadsheets* and a fourth for *word processors*. If we were to place CleverX in this context it would sit right at the centre of these overlapping circles.



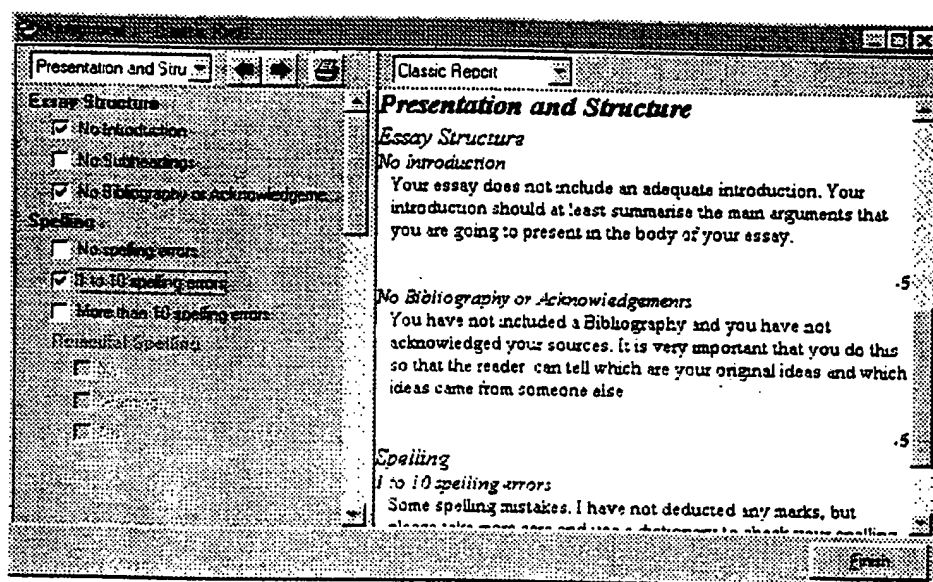
How You Use CleverX

To understand how CleverX works consider the situation of a teacher marking student essays. Suppose that the teacher wants to provide feedback to students about the structure of their essays. One possible comment about the structure might be:

Your essay does not include an adequate introduction. Your introduction should at least summarise the main arguments that you are going to use in the body of your essay.

Generally the teacher will hand write this comment onto the student's paper, or may type the feedback into a report. The teacher may end up writing or typing this feedback many times because a lot of students will make the same mistake.

CleverX provides a much quicker way to give this feedback to students. Using the CleverX Assessment Editor shown below, the teacher simply selects the appropriate comment from a set of displayed comments, or responses that the teacher has previously defined.



Selecting a response on the Assessment Editor causes CleverX to do several things:

- CleverX automatically inserts the comment into a report that can be given to the student
- CleverX automatically updates the student's mark using a score associated with the comment

- CleverX can automatically change the set of responses that are available depending on the responses that have been selected so far. So, for example, if a teacher selects a comment that the student's spelling is poor, new responses can become available for the teacher to indicate how bad the spelling is.

The teacher creates, maintains, and structures the available set of responses by editing a CleverX Project.

Once the teacher has used CleverX to mark all the assignments for a class, the system can be used to determine student marks based on the selected Responses, print all the student's reports in various formats, and collate and analyse data about the whole class.

Phases Involved In Using CleverX

Conceptually there are three phases involved in using CleverX:

- | | |
|------------|--|
| Phase I. | You use CleverX to capture knowledge in a CleverX Project. |
| Phase II. | You use CleverX to complete a task that uses the knowledge in the CleverX Project. |
| Phase III. | You generate reports and perform analysis based on the data entered in Phase II. |

Phase I - Capturing Knowledge in CleverX Projects

Phase I is a *preparation phase* where you build a *knowledge tree* that contains the knowledge for use in Phase II.

You can think of this preparation phase as similar to setting up the template for a form letter without providing the names and addresses of the people to whom the form letter is to be sent. The preparation phase establishes a CleverX template into which specific data can be entered at a later date. This is also similar to creating the formulas in a spreadsheet without actually entering specific figures – the formulas are a template that is used for a specific purpose at a later date.

A very important advantage of CleverX is that it allows knowledge to be easily captured or changed, *at any time*, during any of the phases. You can add, delete and modify information in a CleverX knowledge tree even after you have been using the project.

For example, building a CleverX Project might involve generating a set of marking criteria for an essay topic, associating marks with each of the criteria, and defining feedback comments that should be given to students for each criteria. You could then use this Project to mark some assignments (Phase 2) and then discover some additional criteria that you want to use that you did not include in the Project. CleverX allows you to add these criteria and automatically updates any of the existing assessments that you have created.

Phase II - Using Knowledge in a CleverX Project

Phase II is where CleverX is used to conduct *assessments*. An Assessment is a single use of the knowledge that is contained within the Project. Examples would include an Assessment for a single student's essay, or an Assessment about the condition of a single patient by a doctor.

Obviously a single CleverX Project can be used to create many Assessments. For example, if you are marking student essays then it is here that you would use CleverX to choose the different marking criteria that apply to each of the student's essays. You would generate an Assessment for each student in the class based on a single common set of marking criteria.

You can think of this step as being similar to generating many different invoices based on single spreadsheet, or creating many form letters to different people based on a single common template. In the case of CleverX, you create many new Assessments using the knowledge captured in the Project.

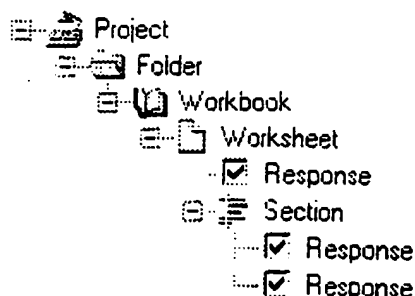
Phase III - Producing Useful Output

Phase III is where CleverX is used to produce additional output and reports based on the outcomes of Phase II. For example, this is where student marks can be totalled, reports giving feedback or comments to the students can be printed, and analysis can be performed on the overall performance of the class. This stage is similar to using a spreadsheet package to generate charts or reports based on data entered into the spreadsheet.

CleverX Components

In CleverX you build Projects out of special CleverX *components*. There are several different types of component, each plays a role in capturing, structuring, and deploying the expert knowledge in the Project. Some components are used to provide different forms of structure by grouping components, while others allow users to enter data when they are using the CleverX Project to create Assessments.

The illustration below shows the structure of components (a knowledge tree) within a CleverX Project. Each of these components are described in the sections that follow.



Components for Structuring Knowledge

Some components are used exclusively to structure the knowledge in a CleverX Project. You use these components to build a *knowledge tree* in which expert knowledge is structured in a hierarchical fashion. These components are

- Folders
- Workbooks
- Worksheets
- Sections

These components are listed in the order that they would appear in a knowledge tree: Folders contain Workbooks, Workbooks contain Worksheets, and Worksheets contain Sections. Each type of component has its own set of properties.

You can think of this knowledge tree structure as being similar to that used by the Windows Explorer. Windows Folders can be used to contain other Windows Folders, Files, and Shortcuts. Similarly CleverX Folders, Workbooks, Worksheets and Sections can be used to contain other CleverX components.

Components for Deploying Knowledge

Some components are used to enter data that is specific to the individual Assessments that you create using CleverX. These components are Responses and Text Boxes.

Responses and Text Boxes are contained within either Sections or Worksheets and they are the "leaves" of the knowledge tree.

CleverX Responses allow you to select the different parts of the knowledge tree that you want to apply to an individual Assessment. By merely pointing and clicking on a particular Response, CleverX will transmit the comment associated with that Response to a report. Because each Response can have an associated score, the selected Responses can also be used to derive numeric values, such as the mark that the student's work should be given, or a price for a quote.

An example of a Response might be "Your essay has more than 10 spelling errors". This Response could be associated with a feedback comment for the student (e.g. "Your spelling needs to be improved. Use a dictionary in future") as well as a score (e.g. automatically deduct 5 marks if this Response is selected).

CleverX Text Boxes allow users to enter text that is specific to an Assessment, as opposed to simply selecting a Response. For example, if you are marking essays then you will probably need to enter the student's name and student number to identify who each of the Assessments applies to. You do this using a CleverX Text Box.

Component Properties

Each type of component in CleverX has a set of *properties* or *attributes*. These properties are used to store the knowledge in the CleverX Project, and the expert who

builds the Project provides values for each of the properties. Different properties apply to the different types of components, but the important properties of CleverX components are described below.

Component Name

A component's name is used on the CleverX Explorer and other screens to identify the component. The name of a component should provide a meaningful indication of the role of the component in the Project. For example a Folder that contains Assessments for student papers on "Romeo and Juliet" should probably be called "Romeo and Juliet Assessments".

Component Comment

CleverX Workbooks, Worksheets, Sections, and Responses can all have *comments* associated with them. The comment provides the detailed expert information that is associated with the component. This information can be sent to reports generated using the Project to conduct assessments

Component Help Text

When you add components to a CleverX Project you can add Help text to the component to indicate to later users of the Project what you intended the component to be used for.

Component Score

Scores only apply to Responses, but they provide the ability to calculate some numeric value for each Assessment based on the Responses that were selected. So, for example, a student's mark, or the price for a quote could be automatically calculated based on the Responses selected for an Assessment.

Relationships Between Components

CleverX includes mechanisms for ensuring that user-defined relationships between components are automatically maintained.

Mandatory Values

The simplest relationship is the ability to insist that at least one Response within a Section must be selected before the system will allow a user to finish with an Assessment.

For example, the creator of a CleverX Project for marking essays may have decided that the user must provide information about whether or not the student's essay was handed in on time or whether the essay was late. A section called "Due Date" might

contain the Responses "On time" and "Late" and CleverX will enforce the fact that one of these must be selected for each Assessment that is created.

Hiding Components

Another constraint that can be applied to component relationships is to make the display of certain Responses contingent on values selected for other Responses. So, for example, a CleverX Project for marking exams might contain marking criteria for each and every question in the exam. When a student sits the exam, however, the student may have the choice of answering either Question 1a or Question 1b, but does not have to answer both. In this situation the CleverX Project would be structured to hide all the components that relate to Question 1a if the student had answered Question 1b.

Maintaining Consistency When Changes are Made

An important feature of CleverX Projects is the way that they accommodate *change*. For example, suppose that midway through marking a set of assignments, you decide that a particular Response needs to be changed. For consistency, you might expect that it would be necessary to go back and alter all the assessments that have been previously made that include the original response that we propose to change. With CleverX this is not necessary. The system automatically, and transparently to the user, updates all the assessments and reports that have been made to accommodate changes to responses, their associated fields and scores. This feature is similar to that found in spreadsheets where, when one changes a number in a column of figures other dependent cells are automatically recomputed to reflect the change.

Generating Reports and Statistics

CleverX's prime reporting capability is to generate individual reports and statistics that relate to each assessment made. Another significant advantage of the CleverX system is that it also has the capability to generate different reports associated with any collection of Assessments or with a whole group or class of assessments. The system makes it very easy to obtain a selection of information that may be used to improve quality in various ways. For example, when a generated report showing the *frequency-of-use* of different responses is examined it may show some responses are used much more frequently than others are in conducting a large batch of assessments. This information could then be used to guide improvement strategies and monitor or compare existing assessments with subsequent assessments.

CHAPTER THREE

YOUR FIRST CLEVERX PROJECT

This chapter guides you through the creation and use of a simple CleverX Project.

You can benefit from this chapter without reading anything else in the User Manual. However, you might find it helpful to return to *How You Use CleverX* in CHAPTER to gain an understanding of important concepts underlying CleverX.

The Project you will create in this chapter is a small part of a Project that a school teacher might use for marking students' essays.

The structure of the Project we are going to create is:

- The Project name is "English Literature". This Project will contain all the assessments for the English Literature class.
- The Project will contain a Folder that we will call "Henry IV". This Folder will hold all the assessments for this text.
- In the "Henry IV" Folder there will be a Workbook called "Assignment I". This Workbook will contain all the student assessments for this assignment.
- The "Assignment I" Workbook would ordinarily contain all the assessment criteria that are to be used for this assignment. For the purposes of building this example we will create only simple criteria about the presentation of the essay. The presentation criteria for this assignment are particularly strict, and we are going to provide comments and deduct marks if the presentation of the essay breaks any of the following rules:
 - The essay must have been handed in on time.
 - The essay must have an introduction that outlines the arguments to be used in the rest of the essay.
 - The student should have made good use of subheadings.
 - The essay must include a Bibliography and acknowledgment of sources.
 - The spelling must be correct.

In addition, if a student makes more than 10 spelling errors, then CleverX will offer the teacher the option of referring the student to remedial spelling lessons.

Building Your First Project

In this section you will build the CleverX Project described above. In later sections of this tutorial you will see how to use the Project to mark some assignments, and then how to generate reports based on this marking.

Starting CleverX

You can start CleverX using any of the standard methods for starting an application under Windows:

- By clicking Start on the Task Bar and then selecting CleverX from the program group that you specified during setup

or

- By using the Windows Explorer to find the CleverX executable file (cleverx.exe) in its program group and then double clicking the CleverX icon

or

- By clicking Start | Run on the Task Bar, typing "cleverx" into the Open text box, and then clicking OK.

or

- By creating a shortcut to the CleverX executable and double clicking this.

Creating and Opening Projects

When CleverX starts, the Welcome screen offers you the choice of opening an existing project or creating a new project.

- Selecting the option to create a new project causes CleverX to create a new blank project for you.
- Selecting the option to open an existing project causes CleverX to display a standard Windows "Open File" dialog box. You can then use this dialog to locate and open the CleverX project that you want.

To start building your first CleverX Project, select the New Project option from the Welcome screen.

Other ways of creating a new project are:

- You can select File | New from the CleverX menu bar

- You can click the New Button on the CleverX Standard toolbar.

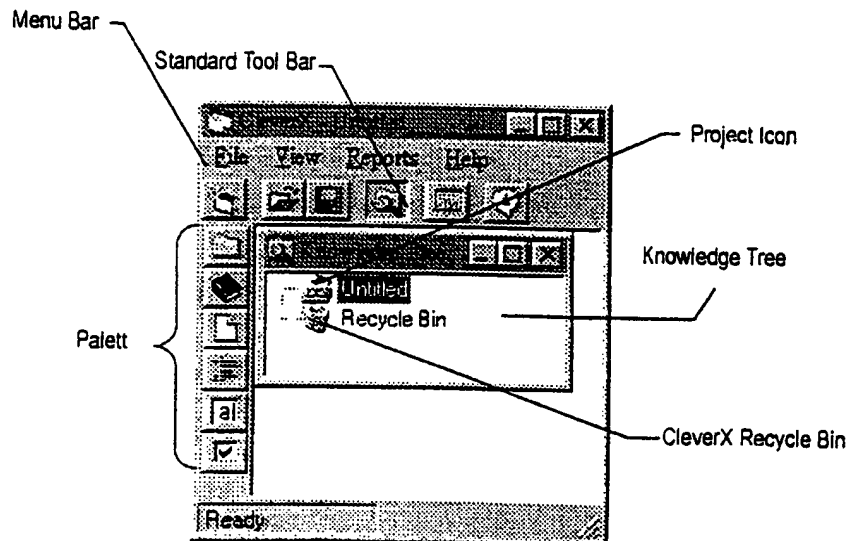
Other ways of opening an existing project are:

- You can select File | Open from the CleverX menu bar
- You can click the Open Button on the CleverX Standard toolbar.

CleverX will only allow you to open a single project at a time. If you try to open a project or create a new project while you have another project open, CleverX will close the current project after prompting you to save.

The CleverX Main Window

After you select the New Project option on the Welcome screen you will see the screen shown below. This screen shows the toolbars and menus that are most often used when creating a CleverX Project.



The Project Icon







The Project Icon represents the new empty Project that CleverX has created for you. You will see how to give the new Project a name in a moment.

The Recycle Bin

The Recycle Bin is the place where all deleted CleverX components are kept before you permanently delete them by emptying the Recycle Bin.

The Standard Toolbar and Menu Items

The Standard Toolbar provides quick access to commonly used CleverX functions as follows:

Toolbar	Menu Item	Function
	File New	Use the New button or menu item to create a new CleverX Project
	File Open	Use the Open button or menu item to open an existing CleverX project file.
	File Save	Use the Save button or menu item to save the currently open Project
	View Project Explorer	Use the Project Explorer button or menu item to show/hide the Project Explorer window
	View Workbook Browser	Use the Workbook Browser button or menu item to show/hide the Workbook Browser window
	View Project Help	Use the Project Help button or menu item to the help that you have supplied for the current project's components.

The CleverX Knowledge Tree

You will use the Knowledge Tree as the main interface for building and maintaining CleverX Projects. The Knowledge Tree allows you to change the structure of the currently open Project and provides the main window for access to the properties of components. We will use the Knowledge Tree in this chapter to create the example Project.

The Tool Pallett

You use the CleverX Tool Pallett to add components to a CleverX Project. Components are the basic elements that are used to build Projects. We will see examples of how components are used in the rest of this chapter.

Saving and Naming a New Project

Once you have created your new Project you should save the Project to a file. You should save regularly to avoid losing data in the event of a power failure, or in case your computer crashes for any reason.

To Save a New Project

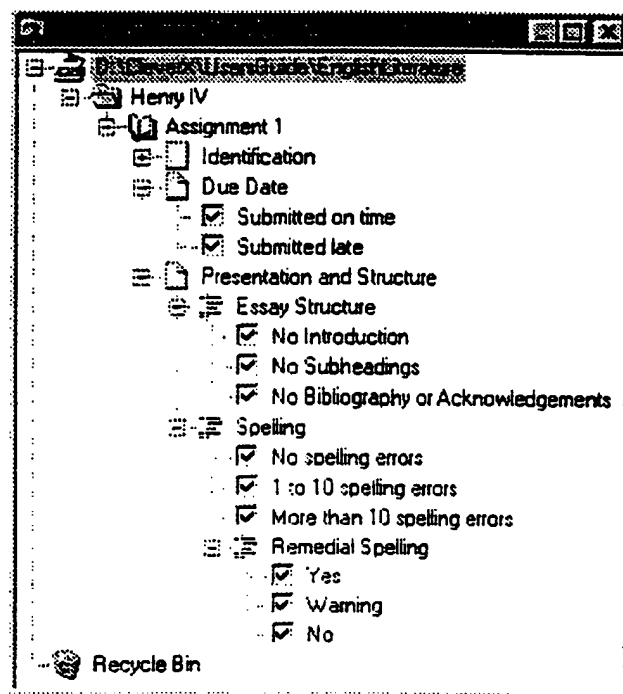
- 1 Click the Save button on the toolbar, or select File | Save from the menu. If you have not saved this project before then you will see a standard Save As dialog box.
- 2 Select a Windows folder from the Save In box, or double-click a different Windows folder in the main window. To create a new Windows folder for the document, click Create New Folder button.
- 3 In the File Name box, type a name for the document. By convention, CleverX files always have a .cx extension automatically appended.
- 4 Click Save.

Save your new project using the file name "English Literature".

Note that once you return to the Project Explorer, the name of your new Project is set to match the name of the file.

Adding and Naming Components

Now that you have created and named a new Project you can add components into the Project. The final version of the Project we are going to build is shown in the picture below.



The example project uses all the different types of CleverX components, but the process for adding and naming components is the same regardless of the type of component involved. Before you start to build the example Project read the procedures for adding and naming components below.

To Add A Component to a Project

- 1 Select the component you want from the CleverX Palette (left-hand vertical column).
- 2 Drag the component from the Palette onto the Project Explorer and drop the new component onto an existing component.

There are some constraints about where components can be dropped, and these are explained later in this Guide. For the moment you do not need to worry about these.

To Rename A Component

When a component is added to a Project, CleverX gives it a default name. You will want to give the component a meaningful name.

- 1 Select the component that you want to rename.

- 2 While the component is selected, right click on the component. A pop-up menu appears. Select Rename.
- 3 A box appears around the name of the component and the text will be highlighted.
- 4 Enter a new name for the component.
- 5 Press Enter.

To Build the Example Project

Constructing the example Project involves adding and naming several different CleverX components. The following instructions will guide you through doing this.



Folder
Button

Add the "Henry IV" Folder

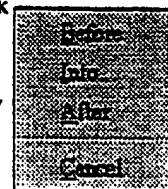
- 1 Use the Folder Button on the Palette to drag and drop a new Folder onto the Project icon. (To "drag and drop" you left-click on icon, hold-down, drag the icon, and release when over the icon it is to be attached to - see the Windows Help for more detailed instructions).
- 2 Rename the new Folder so that it is called "Henry IV"



Workbook
Button

Add the "Assignment I" Workbook

- 3 Now use the Workbook Button on the Palette to drag and drop a new Workbook onto the "Henry IV" Folder.
- 4 A pop-up up menu will appear with the options Before, After and Into. Select the Into option and a new Workbook will be added into the Folder. The "Henry IV" Folder will open to show the new Workbook inside.
- 5 Rename the new Workbook "Assignment I"



Pop-up Menu

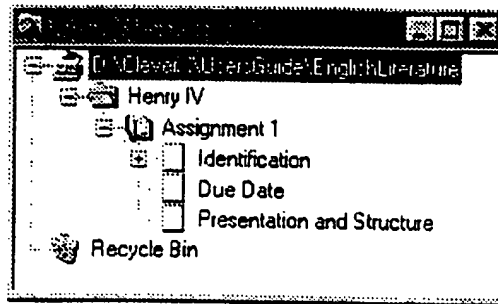


Worksheet
Button

Add Worksheets for "Presentation and Structure" and "Due Date"

- 6 Now use the Worksheet Button on the Palette to drag and drop a new Worksheet into the "Assignment I" Workbook.
- 7 The "Assignment I" Workbook will open to show the new Worksheet inside. You will also see another Worksheet called "Identification". Ignore this for the moment; it will be explained later in this tutorial.
- 8 Rename the new Worksheet "Due Date".
- 9 Now use the Worksheet Button on the Palette to drag and drop another Worksheet into the "Assignment I" Workbook.
- 10 A new Worksheet will be added into the "Assignment I" Workbook above the "Due Date" Worksheet.

- 11 Rename the new Worksheet "Presentation and Structure". At this point your new Project should look like the illustration below.



You can now add Sections into the Worksheets to group the individual assessment criteria that you will add later.

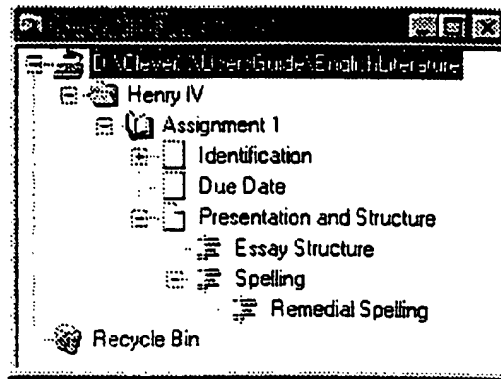
Add Sections for "Essay Structure" and "Spelling"



Section
Button

- 12 Use the Section Button on the Palette to drag and drop a Section into the "Presentation and Structure" Worksheet.
- 13 The "Presentation and Structure" Worksheet will open to show the new Section inside.
- 14 Rename the new Section "Essay Structure".
- 15 Now drag another Section from the Palette and drop it into the "Presentation and Structure" Worksheet.
- 16 Rename the new Section "Spelling".
- 17 Now drag another Section from the Palette and drop it onto the "Spelling" Section.
- 18 A pop-up menu will appear with the options **Before**, **After** and **Into**. Select the **Into** option and a new Section will be added into the "Spelling" Section. The "Spelling" Section will open to show the new Section inside.
- 19 Rename this new Section "Remedial Spelling".

- 20 Now that we have added all the Sections your new Project should look like the illustration below.



You can now add each of the individual assessment criteria into the Sections. A Response component is used for each assessment criterion.

Adding Responses to the "Due Date" Section



- 21 Use the Response Button on the Palette to drag and drop a Response into the "Due Date" Worksheet. Do this again so that there are two Responses in the "Due Date" Worksheet.
- 22 Rename one of the new Responses "Submitted on time." and rename the other "Submitted late."

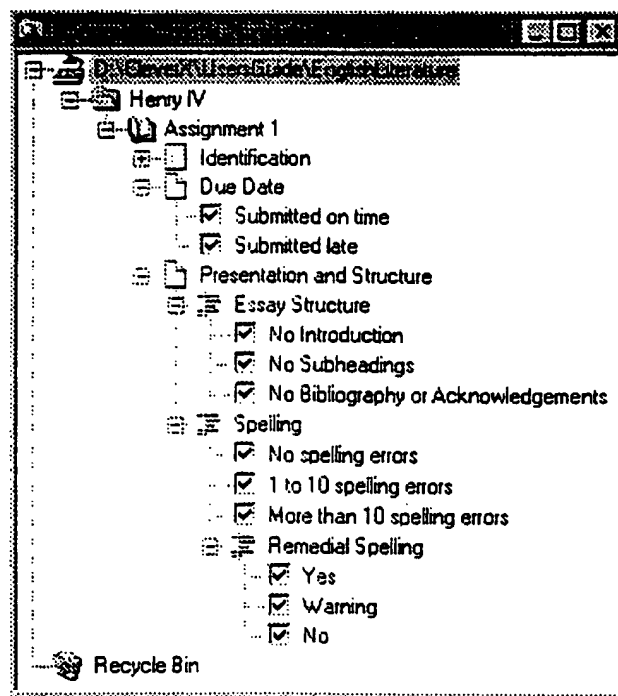
Adding Responses to the "Spelling" Section

- 23 Now add 3 new Responses into the "Spelling" Section.
- 24 Rename these Responses as follows:
- "No spelling errors"
 - "1 to 10 spelling errors"
 - "More than 10 spelling errors"
- 25 Add three Responses into the "Remedial Spelling" section
- 26 Rename these "Yes" and "Warning" and "No"

Adding Responses for "Essay Structure"

- 27 Now add 3 new Responses into the "Essay Structure" Section.
- 28 Rename these Responses as follows:
- "No Introduction"
 - "No Subheadings"
 - "No Bibliography or Acknowledgments"

Once you have done this you have finished adding components to your new Project and it should look like the picture below.



To complete building the Project you need to add comments and scores to the Responses and you need to set some properties of the new components.

Working with the Property Editors

Each component in a CleverX Project has a set of *Properties*. In this section you will see some of the properties of Sections and Responses and will learn how to edit these properties to use several of the more powerful features of CleverX.

Adding Comments and Scores to a Response

Each Response in the sample Project has associated with it some feedback, or comment, that the teacher would like to give to the student if the Response applies to the student's essay. The Response also has associated with it a score that modifies the student's total mark if it is used for the student's work. The table below shows the comments and the scores that you are going to add to the Responses in this part of the tutorial.

Section and Response Name	Comment	Score
Due Date		
On Time	Your essay was handed in on time, thank you.	100
Late	Your essay was handed in late and you have lost 20 marks because of this. If you cannot hand your work in on time please see me before the due date to arrange an extension.	80
Document Layout		
No Introduction	Your essay does not include an adequate introduction. Your introduction should at least summarise the main arguments that you are going to present in the body of your essay.	-5
No Bibliography or Acknowledgments	You have not included a Bibliography and you have not acknowledged your sources. It is very important that you do this so that the reader can tell which are your original ideas and which ideas came from someone else.	-5
No Subheadings	The structure of your essay would be improved substantially if you made use of subheadings. You should use a subheading for your Introduction, Conclusion, and each of the main arguments presented in the body of your paper.	-5
Spelling		
No spelling errors	Excellent spelling. You made no errors.	0
1 to 10 spelling errors	Some spelling mistakes. I have not deducted any marks, but please take more care and use a dictionary to check your spelling before handing in your work next time.	0
More than 10 spelling errors	You have made a lot of spelling mistakes in this essay. This needs improvement - 5 marks have been deducted. Please use a dictionary to check your spelling before handing in your work in future.	-5
Remedial spelling tuition required		
Yes	Your spelling is really not good enough. Please report to Mrs Smith's room on Fridays at 12.30pm for extra spelling drills.	0

Section and Response Name	Comment	Score
Warning	If your spelling does not improve in the near future I will have to consider asking you to attend extra spelling drills.	0
No	No comment given.	0

To add these comments and scores to each of the Responses we use the CleverX Response Property Editor.

To Start the Response Property Editor

- 1 In the Project Explorer use the right mouse button to click on the Response that you want to add a comment and score to.
- 2 A pop-up menu will appear with the options Delete, Rename, Properties, and Cancel. Select Properties and the Response Property Editor will be displayed as shown below.

To Add a Score to a Response

You can use the Response Property Editor to associate a score with a Response. CleverX can then automatically calculate the student's mark, based on the selected Responses.

- 1 In the Response Property Editor for the Response go to the General tab.

- 2 In the Weight field enter the score that you want to associate with this Response. You can enter positive or negative numbers, including decimal numbers.
- 3 Whenever the Response is used in an assessment, CleverX will add the score for the Response onto the total score for the assessment.

To Add a Comment to a Response

You can use the Response Property Editor to associate a comment with a Response. If the Response is selected for a student's essay, CleverX can then automatically send its associated comment to a report on the student's work.

- 1 In the Response Property Editor for the Response go to the Comments tab.
- 2 In the text box enter the comment that you want to associate with this Response.

Note that you can copy and paste text from other sources into the Comments text box. For example, to build the example Project you may want to copy and paste the comments from the table above.

Note that you can also add comments to Sections, Worksheets, and Workbooks although this functionality is not used in the example Project.

Adding Comments and Scores To The Example Project

Now use the information in the table above to add comments and scores to each of the Responses in the Example Project. For each Response you

- 1 Start the Response Property Editor for the Response
- 2 Add the Score
- 3 Add the Comment
- 4 Click Finish

Selecting a Response by Default

When you are conducting an Assessment using CleverX there may be some Responses that you want to select by default for each student. For the example Project, if most students make between 1 and 10 spelling errors, then you may want the "1 to 10 spelling errors" Response in the "Spelling" Worksheet selected by default. This will mean that when you are creating Assessments you will only have to change the selected "Spelling" Response for the essays that have no or many spelling errors.

To Select a Response By Default

- 1 Start the Response Property Editor for the Response you want to select by default.
- 2 In the General tab select the Select by Default check box.
- 3 Click Finished.

In your example Project mark the "1 to 10 spelling errors" Response as selected by default.

Single and Multiple Choice Sections

The "Spelling" Section of the example Project contains a set of Responses that are mutually exclusive – it only makes sense to have one of these Responses selected at any one time because only one of the Responses can apply to any single essay.

On the other hand, any number of the Responses in the "Essay Structure" Section could be selected for the same essay.

CleverX automates this concept by allowing you to set a property on a Section saying whether one or many Responses can be selected in the Section.

By default, when CleverX creates a new Section, only one Response in that Section can be selected. To change this property you use the Section Property Editor.

To Start the Section Property Editor

- 1 In the Project Explorer use the right mouse button to click on the Section that you want to change the properties for.
- 2 A pop-up menu will appear with the options Delete, Properties, and Cancel. Select Properties and the Section Property Editor will be displayed. The Section Property Editor is very similar in appearance to the Response Property Editor.

To Set The Number of Responses that can be Selected in a Section

- 1 Start the Section Property Editor for the Section you want.
- 2 In the General tab select the desired radio button under the Response Exclusivity heading.

You will want to set the "Essay Structure" Section of the example Project to allow multiple components to be selected. Do this now.

Hiding Components

In the example Project the Section "Remedial Spelling" only needs to be visible if the student's essay contains more than 10 spelling errors. If the student made 10 errors or less then the teacher does not want to be bothered by seeing these components on the screen. If the student's essay contained more than 10 errors, however, the teacher wants to be reminded about the option to send the student to remedial classes.

CleverX makes this possible by allowing you to hide components depending on the values selected for other Responses. There are two ways this can happen:

- You can hide a set of components whenever a particular response is selected
- You can hide a set of components whenever a particular response is *not* selected

In the example Project you can use either approach to hide the "Remedial Spelling" section. The first approach is illustrated below for the example Project. The general procedure for hiding a component is as follows:

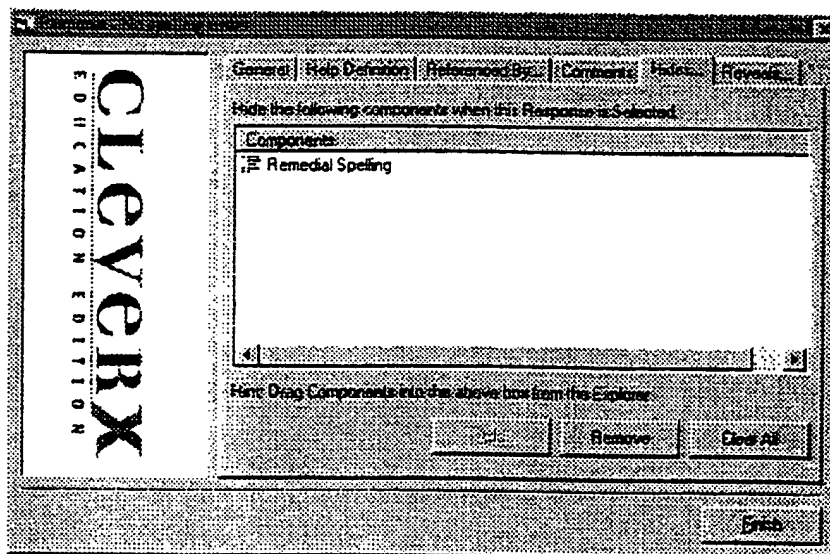
To Hide A Component

- 1 Start the Response Property Editor for the Response that you want to use to hide other components.
- 2 Arrange your CleverX windows so that you can see both the Response Property Editor and the Project Explorer
- 3 In the Response Property Editor go to the Excludes... tab
- 4 From the Project Explorer drag the component that you want to hide onto the Response Property Editor and drop it into one of the Object Windows. If you want the component to be hidden when the Response is selected then use the left hand Object window. If you want the component to be hidden when the Response is not selected then use the right hand Object window
- 5 Click Finished.

Hiding the "Remedial Tuition" Section

The "Remedial Tuition" Section should be hidden if either of the Responses "No spelling errors" or "1 to 10 spelling errors" are selected. To do this you set both of these components to hide the "Remedial Tuition" Section if they are selected.

- 1 Start the Response Property Editor for the "No spelling errors" Response
- 2 Arrange your CleverX windows so that you can see both the Response Property Editor and the Knowledge Tree.
- 3 In the Response Property Editor go to the Hides... tab.



- 4 In the Knowledge Tree select the "Remedial Spelling" component.
- 5 In the Response Property Editor click the Add button. The Response Property Editor window should then appear as above.
- 6 Click Finished on the Response Property Editor.
- 7 Repeat this process for the "1 to 10 spelling errors" Response

In the next part of this tutorial you will start using the example Project to mark some essays, and then you will see how CleverX hides the "Remedial Tuition" component when creating Assessments.

Working with Your New Project

The first part of this chapter has shown how to build a CleverX Project. This section now shows you how to use this Project to simplify the marking of student's essays.

When you use a CleverX Project you create *Assessments*. An assessment is simply the result of selecting Project Responses to create a single report. Try using CleverX to mark a single student's essay. As you mark a large pile of essays you create one Assessment for each essay.

When you created the example project you did all of the work using the Project Explorer and the Property Editors. When you are using a Project you do most of your work in the Workbook Browser and Assessment Editor. These windows are explained in the sections that follow.

The CleverX Workbook Browser

The Workbook Browser allows you to work with Assessments for a single Workbook. You can see a list of all the Assessments you have created using the Workbook, create new Assessments, open existing Assessments and delete Assessments.

The example Project contains one Workbook for "Assignment 1". In this part of the tutorial you will use this Workbook to mark some hypothetical essays.

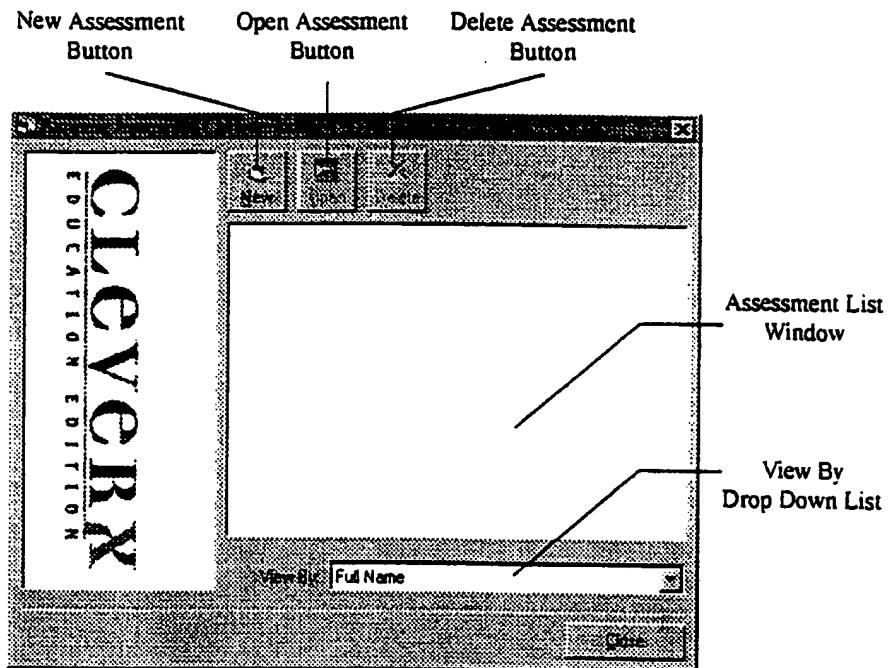
To Start the Workbook Browser

- 1 In the Project Explorer select the Workbook that you want to work with - "Assignment 1" by clicking on it.
- 2 The Workbook Browser Button on the Toolbar becomes active when a Workbook is selected. Click the Workbook Browser Button.
- 3 The CleverX Workbook Browser is displayed.



Workbook
Browser
Button

After you have opened the Workbook Browser for "Assignment 1" you will see the screen below.



The Assessment List Window

The Assessment List Window shows a list of all the Assessments that exist for this Workbook. For the example Project this window will list all the students for which an Assessment has been created. When you first open the Workbook Browser this list will be empty because you have not yet created any Assessments for "Assignment 1"




The "View By" Drop Down List

You use this pull down menu to choose how the assessments in the Assessment List window are identified. For the example Project you can identify student assessments using any of the following:

- The student's full name
- The students first name only
- The student's last name only
- The student's student number.

The Workbook Browser Buttons

The Workbook Browser has buttons that allow you to work with Assessments as follows:

Button	Function
	Use the New button to create a new Assessment using the current Workbook.
	Use the Open button to open the Assessment that is currently selected in the Assessment List Window
	Use the Delete button to delete the Assessment that is currently selected in the Assessment List Window

The CleverX Assessment Editor

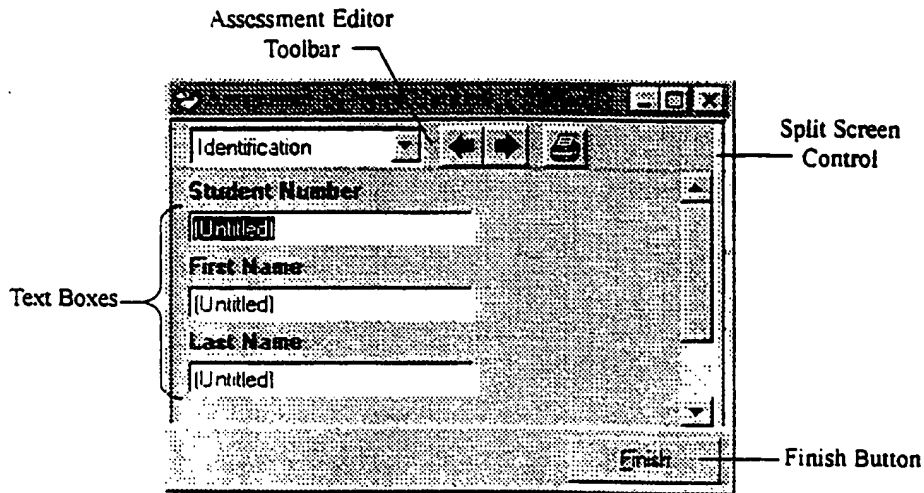
The CleverX Assessment Editor is the screen in which you can enter data for a particular Assessment. This screen allows you to select the Responses that apply to an Assessment and to enter data into Text Boxes for the Assessment.

The Assessment Editor displays a single Worksheet at a time, and you can select different Worksheets using buttons on the Assessment Editor Toolbar.

Starting the Assessment Editor

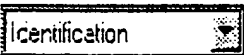



To start the Assessment Editor by creating a new Assessment, click the New Button on the Workbook Browser. For the "Assignment I" Workbook you will see the CleverX Assessment Editor shown below. This Assessment Editor is displaying the Identification Worksheet that CleverX adds to each Workbook when the Workbook is created.

The Identification Worksheet allows you to enter information that identifies the new Assessment. CleverX provides three text boxes named "Student Number", "First Name", and "Last Name". If you want to, you can change the names of these text boxes by renaming them in the Knowledge Tree (see To Rename A Component earlier in this chapter), but the values in these three text boxes are always used to identify the Assessment.



The Assessment Editor Toolbar Buttons

The Assessment Editor has buttons that allow you to work with assessments as follows:

Button	Function
	Use the Worksheet drop down menu to select the Worksheet that you want to display.
	Use the Previous button to move to the previous Worksheet
	Use the Next button to move to the next Worksheet
	Use the Print button to print the Assessment. Note that all the Worksheets in the Workbook are printed.

The Finish Button

Once you have completed an Assessment you close that Assessment by using the Finish button on bottom right hand corner of the Assessment Editor. CleverX automatically saves the Assessment for you.

Text Boxes, Sections, and Responses

The Assessment Editor displays the names of the Sections, Responses, and Text Boxes that are contained within the Worksheet that you are currently working with. Sections are displayed as bold headings; Responses are labelled checkboxes; and Text Boxes are displayed as labelled text entry fields.

Moving Between Worksheets


The Assessment Editor displays a single Worksheet at a time. To move to a different Worksheet you can either

- Use the Previous and Next Buttons on the Assessment Editor Toolbar

Or

- Select the Worksheet you want to move to from the drop down menu on the Assessment Editor Toolbar.

Move to the "Presentation and Structure" Worksheet by clicking on the Worksheet drop down menu and selecting the "Presentation and Structure" Worksheet. The Assessment Editor will then display the Worksheet shown below.

Identification	
Identification	
Due Date	
Presentation and Structure	

Assessment Editor
Toolbar

Presentation and Stru

Essay Structure

☐ No Introduction

☐ No Subheadings

☐ No Bibliography or Acknowledgements

Spelling

☐ No spelling errors

☐ 1 to 10 spelling errors

☐ More than 10 spelling errors

Remedial Spelling

☐ Yes

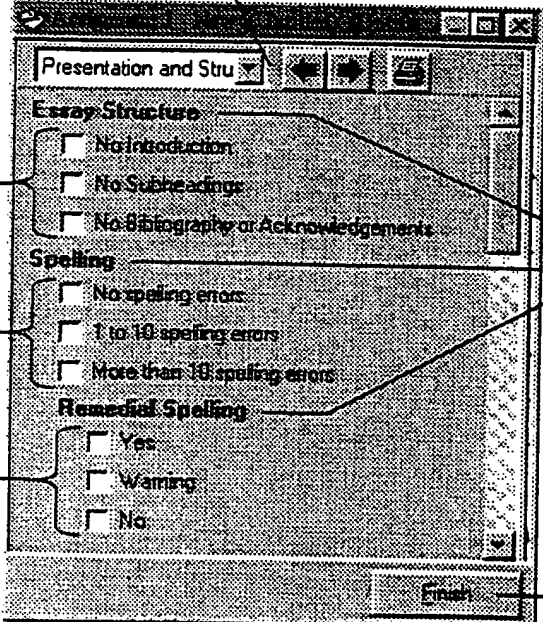
☐ Warning

☐ No

Responses

Sections

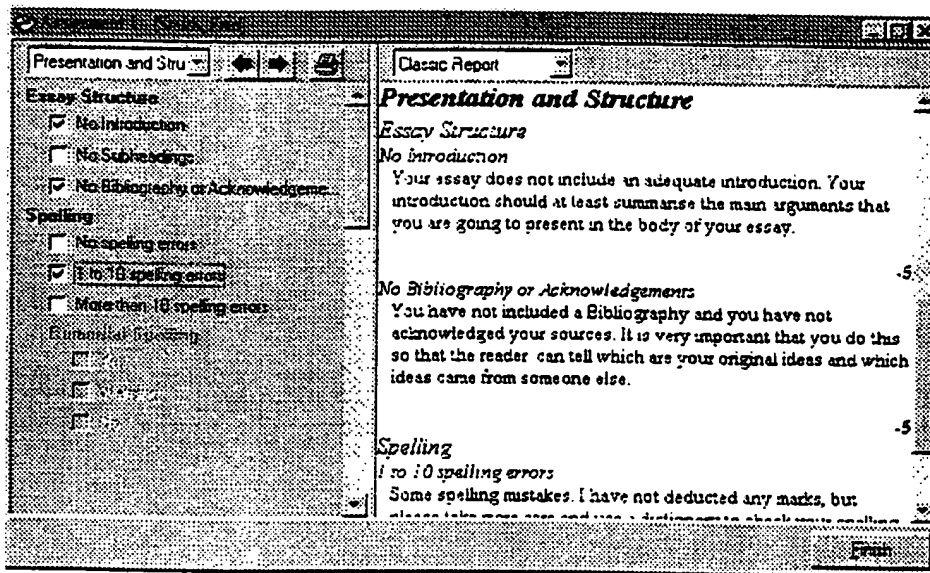
Finish Button



The Split Screen Control

The Assessment Editor has a Split Screen that allows you to view the CleverX Report for the Assessment as you work. To view the CleverX Report on the Assessment Editor:

- 1 Move the cursor over the Split Screen Control until the pointer changes to the Split Screen Cursor (see right)
- 2 Double click and the window will divide to reveal the CleverX Report as seen below. You may have to resize the Assessment Editor window to achieve the window proportions that you want. You can change the amount of the Assessment Editor screen that is devoted to the Report by using the Split Screen Cursor to drag the Split Screen control to the left or right.



Conducting An Assessment

Completing an Assessment with CleverX is the simplest part of using the system. All you do is select the Responses that apply to the Assessment you are creating, and enter data into the Text Boxes for the Assessment.

To Select A Response

- 1 Use the left mouse button to click on the Response Name or the Check Box beside the Response.
- 2 Selected Responses appear ticked ☒

- 3 Unselected responses are blank.



To Enter Data into a Text Box

- 1 Left-click in the Text Box to put the cursor into the Text Box.
- 2 Type the text you want.

Creating Example Assessments

Once you have started the Assessment Editor, you can enter data for the Assessment that you are creating. For the purposes of the tutorial, you can enter data for an assessment as described below. If you have the Split Screen open to reveal the CleverX Report then you will see this report update dynamically as you enter data into the Assessment Editor.

Assessment for Fred Smith

The essay submitted by Fred Smith had the following characteristics:

- There was no introduction
- There was no use of subheadings
- There were more than 10 spelling errors
- You have decided to give the student a warning that he may be sent to remedial spelling classes
- The essay was late

The following procedure explains how to create a CleverX Assessment to reflect these characteristics.

- 1 Move back to the "Identification" Worksheet using the Worksheet drop down menu.
- 2 In the "Student Number" Text Box enter 11111
- 3 In the "First Name" Text Box enter Fred
- 4 In the "Last Name" Text Box enter Smith
- 5 Move to the "Due Date" Worksheet by using the Next Button or by selecting "Due Date" from the drop down menu on the Assessment Editor Toolbar.
- 6 Select the "Submitted Late" Response.
- 7 Move to the "Presentation and Structure" Worksheet by using the Next Button or by selecting "Presentation and Structure" from the drop down menu on the Assessment Editor Toolbar.
- 8 Select the Responses for "No Introduction" and "No Subheadings" in the "Essay Structure Section."

- 9 Select the Response for "More than 10 spelling errors" in the Spelling Section. Note that the Response "1 to 10 spelling errors" becomes unselected automatically and that the Section "Remedial Spelling" appears automatically.
- 10 In the "Remedial Spelling" Section select the "Warning" Response.
- 11 You have now finished creating the Assessment for Fred Smith so click on the Close Button on the top right hand corner of the Assessment Editor.
- 12 The Assessment Editor will close and you will be able to see the new Assessment for Fred Smith in the Worksheet Browser.

CleverX Reports

CleverX allows you to view and print reports for the Assessments that you create. You can customise these reports by:

- selecting which CleverX components are to be included in the report
- selecting which report format is to be used to display or print the report

This section shows you how to work with CleverX Reports.

Printing CleverX Reports

If you have the CleverX Assessment Editor open for an Assessment, you can print the report for that assessment, either to a printer or to a file.

To Print A Report to a Printer

- 1 Open the Assessment Editor for the assessment that you want to print.
- 2 From the Assessment Editor Toolbar select the Print button.
- 3 To print the report to a printer, make sure that the **Print to File** check box is *not* checked.
- 4 Make sure that your printer **Properties** are set correctly.
- 5 Select the number of copies that you want to print.
- 6 Click OK.



Print Button

CleverX also allows you to print reports to a file in Rich Text Format. Rich Text Format can be read by most word processors and so this gives you a way to further edit and format your reports after CleverX has generated them.

To Print a Report To A File

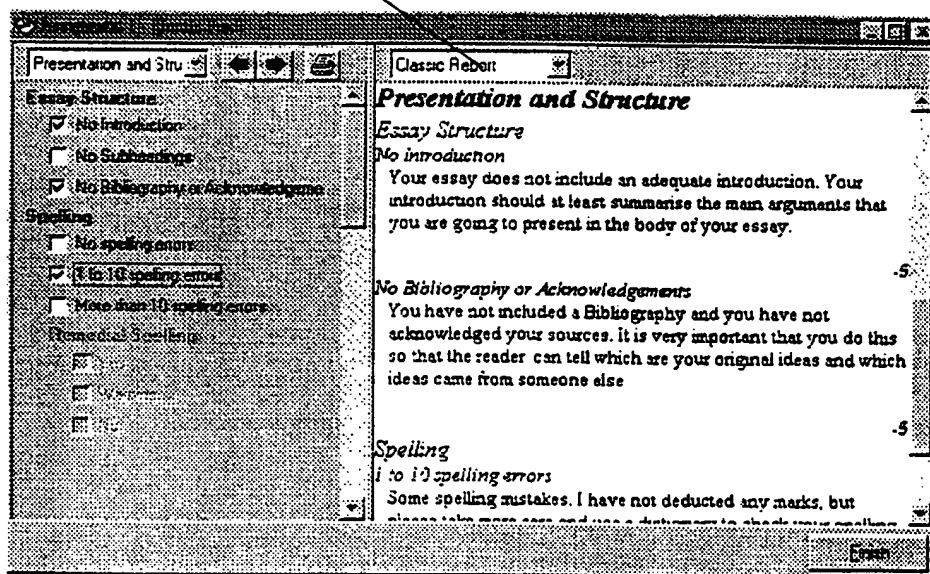
- 1 Go to the Assessment Editor for the assessment that you want to print.
- 2 From the Assessment Editor Toolbar select the Print button.
- 3 Select the **Print to File** check box.

- 4 Click OK
- 5 In the Save As... window select a file name and folder for the new file.
- 6 Click Save

Changing the Selected Report Format

The Assessment Editor Split Screen has a drop down menu in the toolbar that allows you to select the format of the report to use when displaying and printing the report. This menu is shown below. To change the selected report format, click on this drop down menu and select a report format. The split screen window will be automatically updated with the new selection.

Report Selection
Drop Down Menu



The CleverX Report Manager

By default, CleverX reports contain all the information that is available for the current Assessment. This information includes:

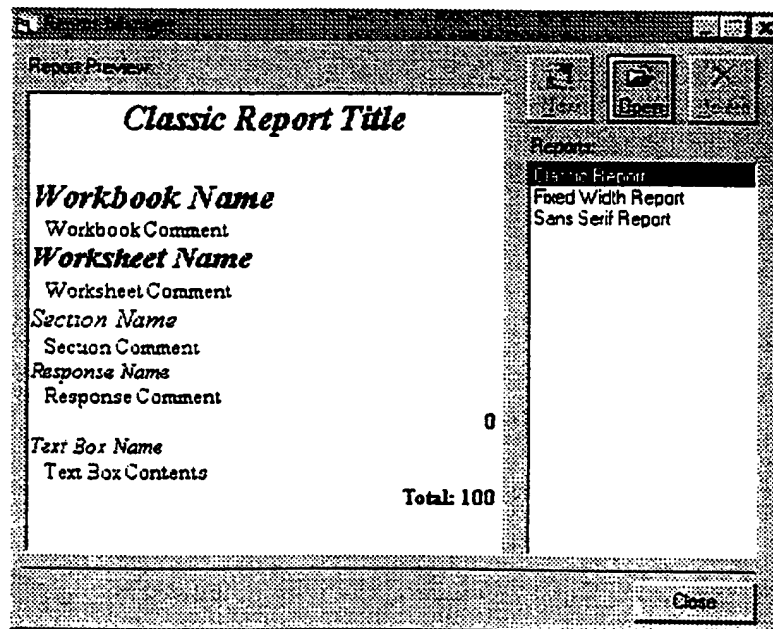
- A title based on the Workbook name and the values in the Text Boxes on the "Identification" Worksheet
- The Workbook name and any comments that have been entered for the Workbook

- The names of all the Worksheets and any comments that have been entered for the Worksheets
- The names of all the Sections and any comments that have been entered for the Sections
- The names of all the selected Responses, any comments that have been entered for the selected Responses, and the score associated with any selected Response.
- The names of all the Text Boxes and any text that has been entered into the Text Boxes
- The total score for the Assessment.

You can customise a CleverX Report to remove any of these pieces of information by using the CleverX Report Manager and the CleverX Report Definition Property Editor.

Starting The Report Manager

You start the Report Manager by selecting Reports | Report Manager from the CleverX main menu. The Report Manager screen is shown below:



The Report Manager includes a list box on the right hand side that shows all the *Report Definitions* that are currently in the system. A Report Definition specifies

what information is included in a report, as well as the text styles that should be used in the report.

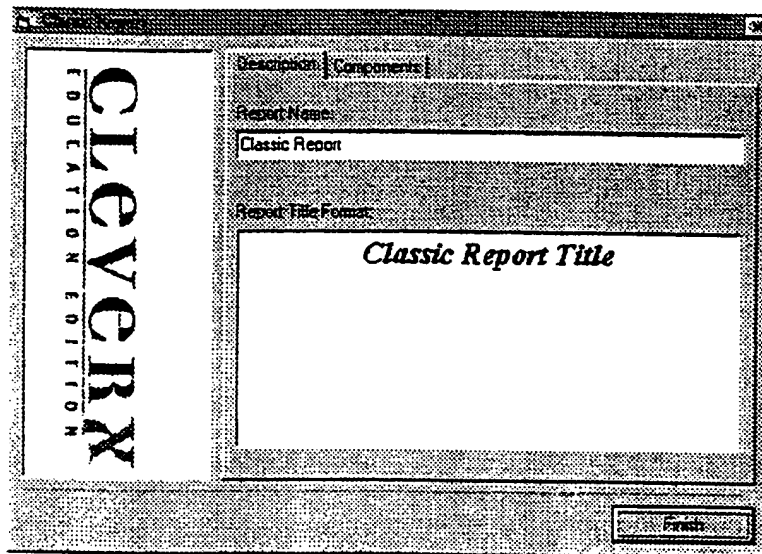
The Report Manager displays a preview of the selected Report Definition and provides buttons to create New Report Definitions, Open the selected Report Definition and Delete a Report Definition. The New and Delete buttons have been disabled for the current release of CleverX and will be made available in a subsequent release.

To customise a particular Report Definition you select the name of the Report Definition and click the Open button on the Report Manager. When you do this the Report Definition Property Editor is displayed.

The Report Definition Property Editor

The Report Definition Property Editor allows you to customise the properties of a Report Definition. When you change a Report Definition in its Property Editor, the changes apply to every Assessment that uses that Report Definition.

The Report Definition Property Editor is shown in the screen below:

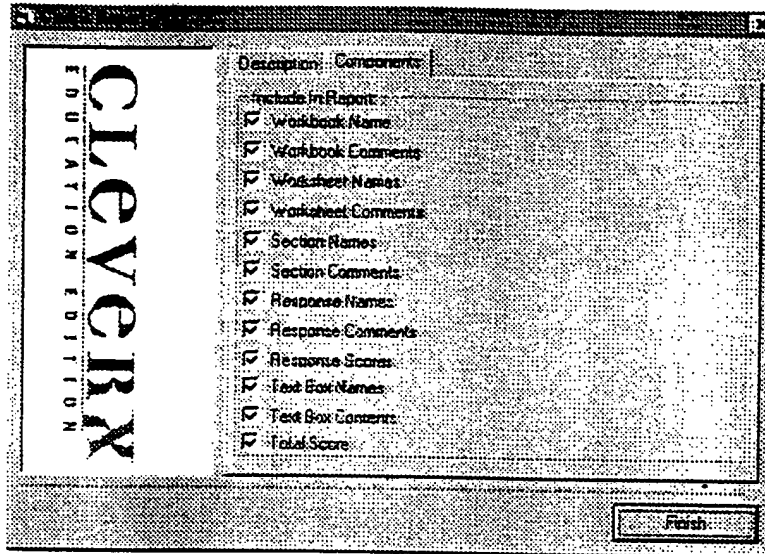


The Description Tab

The Description Tab on the Report Definition Property Editor shows the name of the Report Definition and the style of the text that will be used for the title of the report. At present there is no means of changing the title, however you can rename the Report Definition if you wish.

The Components Tab

The Components Tab on the Report Definition Property Editor allows you to select what information is sent to the CleverX Reports that are based on this Report Definition. The Components Tab is shown below:



To remove pieces of information from this Report Definition simply clear the check box for the information you want to remove. For example, if you only want the comments for a Response to appear when the Response is selected and you do not want the Response name or score, then clear the check boxes marked Response Names and Response Scores.

THE CLAIMS DEFINING THIS INVENTION ARE AS FOLLOWS:-

1. A method of retrievably storing in a computer database, aggregates each consisting of related titled information, the method including:-
 - 5 storing each title as a conventional record in a conventional database so that titles can be efficiently added, moved or removed as conventional records;
providing further database means in which individually titled information of an aggregate may be stored as a
10 conventional record in a further conventional database, and
operatively associating each individually titled information with its title and aggregate whereby each aggregate may be stored, removed, updated or queried.
2. A method as claimed in claim 1 and wherein the
15 aggregates are conceptually organised into rows of aggregates forming columns of related titled information as per a conventional database and wherein said conventional record is of the type in which information in cells containing values is organised into rows of records and columns representing
20 similar types of cell values.
3. A method as claimed in claim 2, wherein said information for each of the columns includes a column's name and any defining characteristic of the column.
4. A method as claimed in any one of the preceding claims,
25 wherein said information for at least a plurality of said column definitions is stored in a separate database.
5. A method as claimed in claim 4, wherein operative association of said field definitions is provided by linking.

6. A method as claimed in claim 4, wherein operative association of cell values is provided by additionally storing a reference to the appropriate column definition along with the respective row for the cell value.

5 7. A method as claimed in claim 4, wherein operative association each cell value with its column is achieved through the use of conventional database foreign keys.

8. A method as claimed in any one of the preceding claims, wherein:-

10 said conventional database table is established as a Column Definition Table and stores the defining characteristics of a column including a column's name and other associated information pertaining to the definition of a column;

15 said further conventional database table is established as a Cell Repository Table storing a column of cell values and a column of associated row values;

association of individual aggregate information with its title and aggregate is achieved by associating individual
20 cell values stored in a Cell Repository Table with their respective row and column values from the Column Definition Table.

9. A method as claimed in claim 8, wherein said Cell Repository Table stores a relatively small number of related
25 cell values, or columns of cells values of the same type.

10. A method of storing information in conventional databases, the method including:-

storing each column definition as a conventional record in a conventional database so that column definitions can be
30 efficiently added, moved or removed as conventional records;
providing further database means in which cell values of

a row are stored as separate records, and
associating each cell value with its column and row.

11. A method of storing in conventional databases
information organised into rows and columns as per a
5 conventional database, the method including:-
storing each column definition as a conventional record
in a conventional database so that column definitions can be
efficiently added, moved or removed as conventional records;
providing further database means in which cell values of
10 a record are stored as separate records, and
associating each cell value with its column.

12. A database structure including:-
a conventional database table established as a
Column Definition Table for storing the defining
15 characteristics of a column including a column's name
and other associated information pertaining to the
definition of a column;
a further conventional database table established as
a Cell Repository Table storing a column of cell values
20 and a column of associated row values, and
associating means for association individual cell
values stored in a Cell Repository Table with their r
espective row and column values from the Column
Definition Table.

1/25

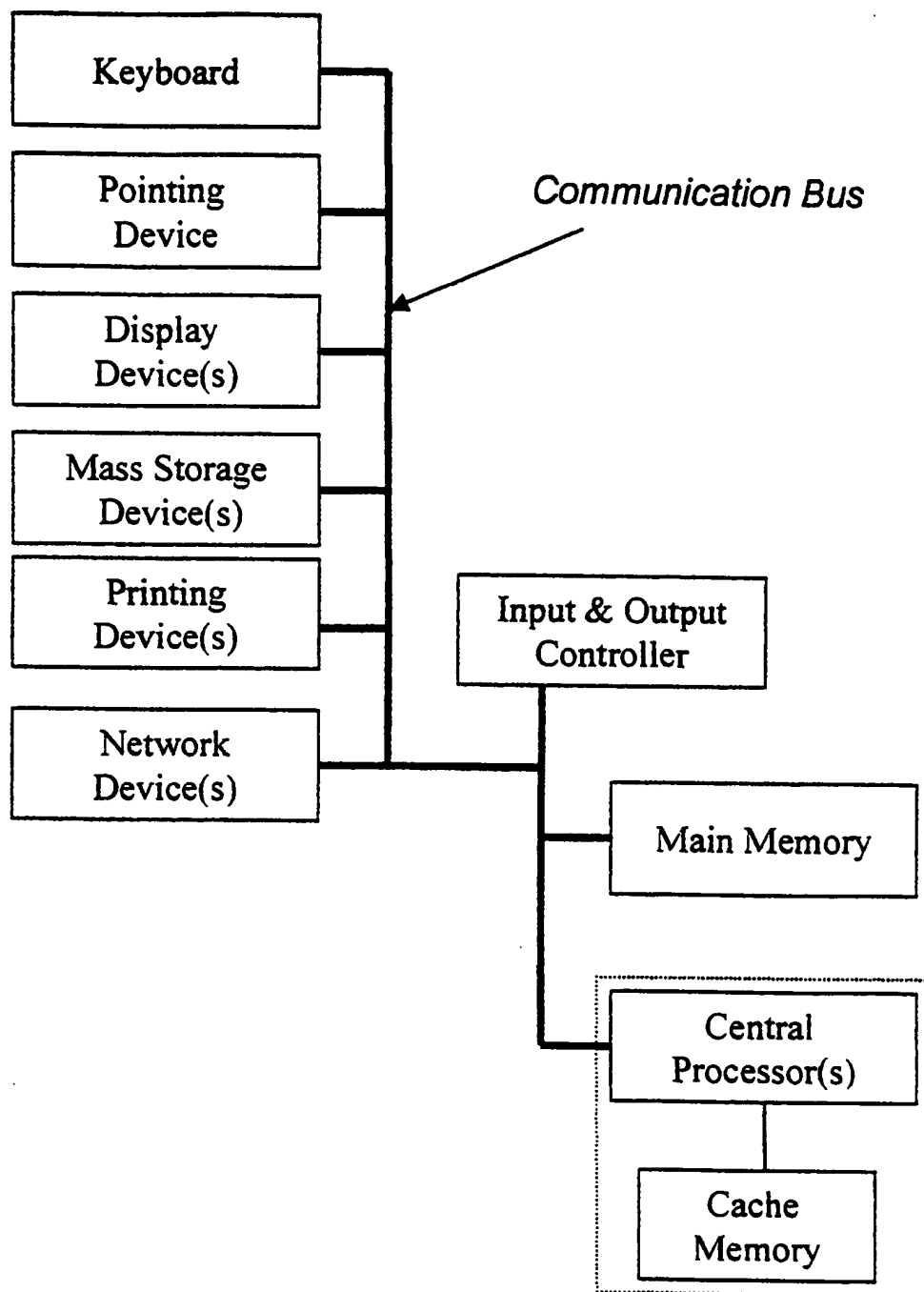


Fig. 1

2/25

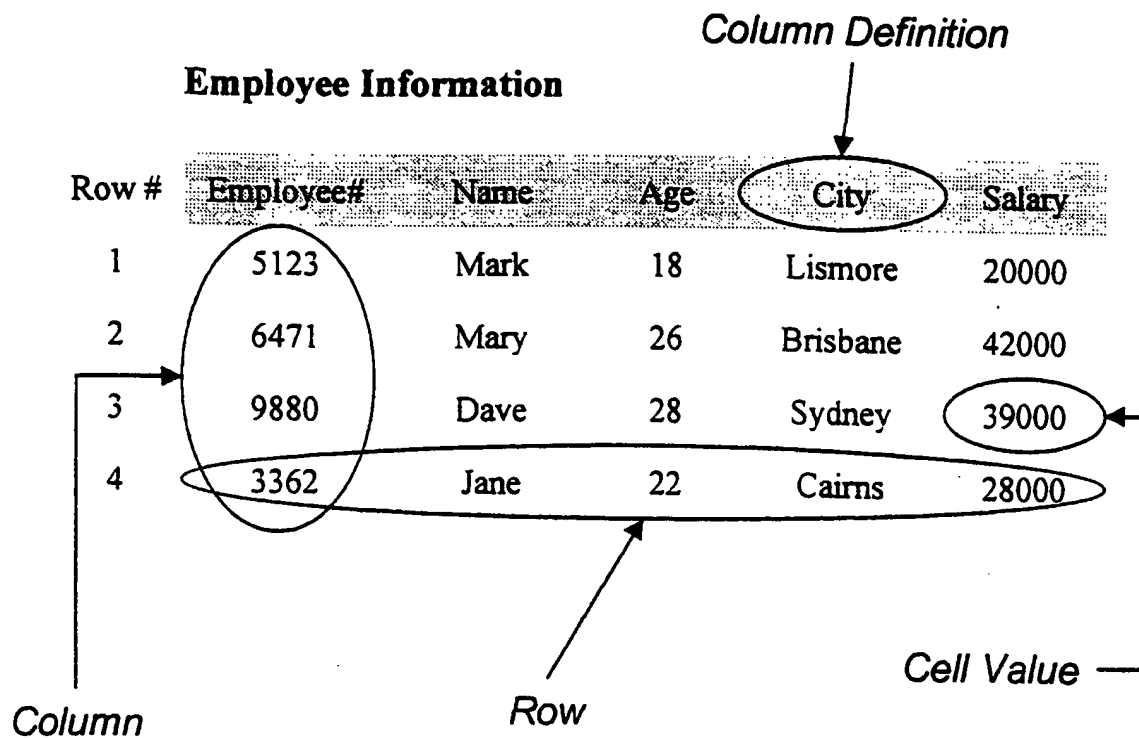


Fig. 2

3/25

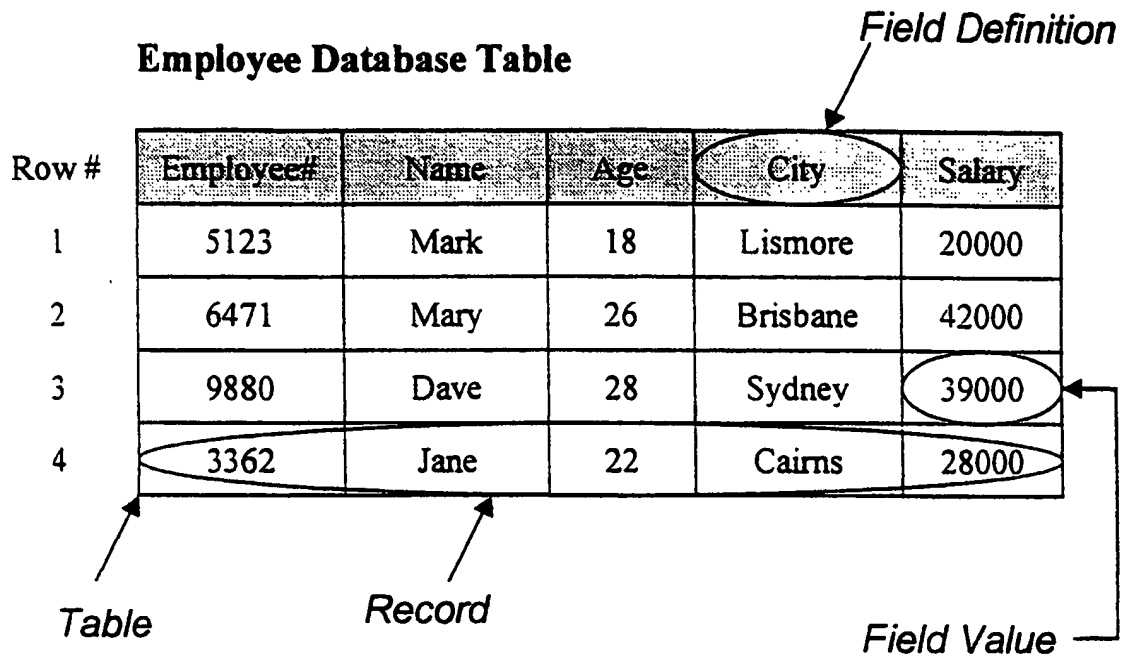


Fig. 3

4/25

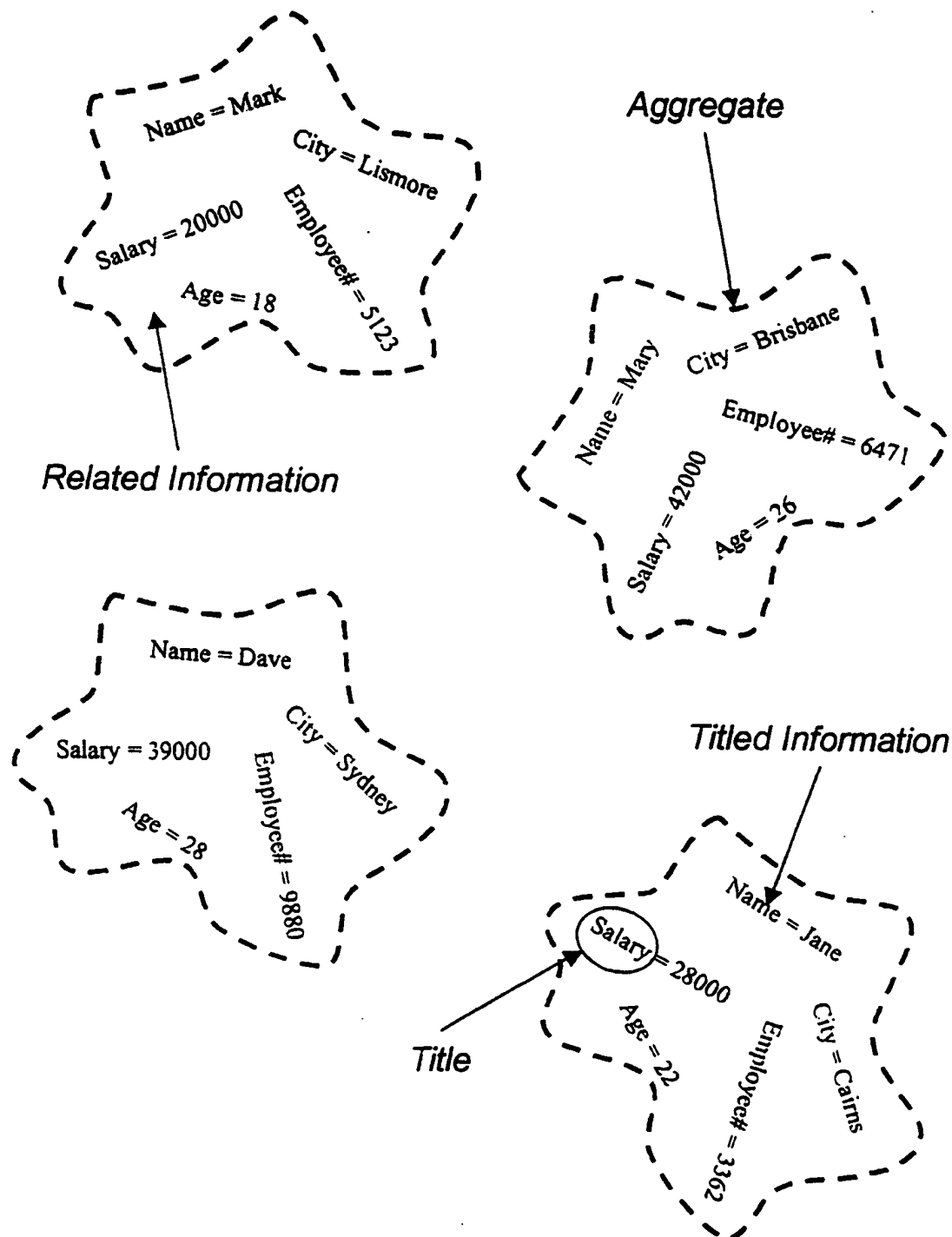


Fig. 4

5/25

Figure 5 Continued

Figure @2.2: String Cell Repository Table

Row#	String ColumnID	Value
1	1	Mark
1	2	Lisnore
2	1	Mary
2	2	Brisbane
3	1	Dave
3	2	Sydney
4	1	Jane
4	2	Cairns

Figure @2.4: String Column Definition Table

ColumnID	Column Name	Length
1	Name	40
2	City	20

Fig. 5 (continued)

Figure 5

Figure @2.1: Numeric Cell Repository Table

Row#	Numeric ColumnID	Value
1	1	5123
1	2	18
1	3	20000
2	1	6471
2	2	26
2	3	42000
3	1	9880
3	2	28
3	3	39000
4	1	3362
4	2	22
4	3	28000

Figure @2.3: Numeric Column Definition Table

ColumnID	Column Name	Maximum
1	Employee#	9999
2	Age	150
3	Salary	100000

Fig. 5

6/25

Original Employee Database Table

Row #	Employee#	Name	Age	City	Salary
1	5123	Mark	18	Lismore	20000
2	6471	Mary	26	Brisbane	42000
3	9880	Dave	28	Sydney	39000
4	3362	Jane	22	Cairns	28000

Copying previous records into the new table containing the new field(s)

New Employee Database Table

Row #	Employee#	Name	Age	City	Salary	Height
1	5123	Mark	18	Lismore	20000	?
2	6471	Mary	26	Brisbane	42000	?
3	9880	Dave	28	Sydney	39000	?
4	3362	Jane	22	Cairns	28000	?

Initial value of the new Field may be undefined, or a specified default

Fig. 6
SUBSTITUTE SHEET (RULE 26)

7/25

Algorithm: Adding a new Column**Input(s):**

cn = Column Name
cp = Additional Column Parameters

Output(s):

id = unique identifier of the new column

Algorithm:

1. Find the Column Definition Table (cdt) for the type of column for the required column cn.
2. Create a new record in cdt where id becomes the new record identifier
3. Store in the new record, the column identifier id, the column name cn, and any additional column parameters cp

Fig. 7

8/25

Numeric Column Definition Table

ColumnID	Column Name	Maximum
1	Employee#	9999
2	Age	150
3	Salary	100000
4	Height	200



New Column Definition

Fig. 8

9/25

Algorithm: Deleting an existing Column**Input(s):**

id = Unique Column Identifier

Output(s):

none

Algorithm:

1. Find the Column Definition Table (cdt) for the type of column specified by the column id
2. Delete the column definition from the cdt for the column with the specified id
3. Delete from the Column Definition Table (cdt) associated Cell Repository Table (crt) cell values that are for a column with the specified id

Fig. 9

10/25

Numeric Cell Repository Table

Row#	Numeric ColumnID	Value
1	1	5123
1	2	18
2	1	6471
2	2	26
3	1	9880
3	2	28
4	1	3362
4	2	22

Numeric Column Definition Table

ColumnID	Column Name	Maximum
1	Employee#	9999
2	Age	150

Fig. 10

11/25

Numeric Cell Repository Table

Row#	Numeric ColumnID	Value
1	1	5123
1	2	18
1	3	20000
2	1	6471
2	2	26
2	3	42000
3	1	9880
3	2	28
3	3	39000
4	1	3362
4	2	22
4	3	28000

*Cell Values for renamed
Column remain unchanged*

Numeric Column Definition Table

ColumnID	Column Name	Maximum
1	Employee#	9999
2	Years Since Birth	150
3	Salary	100000

Renaming a Column

Fig. 11

12/25

Employee Database Table

Row #	Employee#	Name	Age	City	Salary	Partner
1	5123	Mark	18	Lismore	20000	Sally
2	6471	Mary	26	Brisbane	42000	
3	9880	Dave	28	Sydney	39000	
4	3362	Jane	22	Cairns	28000	Fred

*Storage space reserved for a
partners name but not used*

Fig. 12

13/25

String Cell Repository Table

Row#	String ColumnID	Value
1	1	Mark
1	2	Lismore
1	3	Sally
2	1	Mary
2	2	Brisbane
3	1	Dave
3	2	Sydney
4	1	Jane
4	2	Cairns
4	3	Fred

*Only complete/defined
cell values are stored
for the "Partner"
Column*

String Column Definition Table

ColumnID	Column Name	Length
1	Name	40
2	City	20
3	Partner	40

*Definition of the
"Partner" Column*

Fig. 13

14/25

Algorithm: Adding a Row**Input(s):**

r = row number to add
row = set of (ColumnID, CellValue) pairs

Output(s):

none

Algorithm:

1. FOR EACH (col, val) in row set
 - 1.1 Find the appropriate Cell Repository Table (crt) for the column col
 - 1.2 ADD RECORD TO crt WHEREBY THE RECORD FIELD VALUES
Row# becomes r,
ColumnID becomes col, and the
Value becomes val

Fig. 14

String Cell Repository Table

Row#	String ColumnID	Value
1	1	Mark
1	2	Lismore
2	1	Mary
2	2	Brisbane
3	1	Dave
3	2	Sydney
4	1	Jane
4	2	Cairns
5	1	Kevin
5	2	Melbourne

Newly Stored Cell Values

Fig. 15 (continued)

Numeric Cell Repository Table

Row#	Numeric ColumnID	Value
1	1	5123
1	2	18
1	3	20000
2	1	6471
2	2	26
2	3	42000
3	1	9880
3	2	28
3	3	39000
4	1	3362
4	2	22
4	3	28000
5	1	1234
5	2	31
5	3	52000

Newly Stored Cell Values

Fig. 15

16/25

Algorithm: Retrieving a Row**Input(s):**

r = row number to retrieve

Output(s):

row = set of (ColumnID, CellValue) pairs

Algorithm:

1. row = { }
2. FOR EACH Cell Repository Table (crt) in the Database
for row r
 - 2.1 Seek to row r in crt
 - 2.2 Add to row set the pair
(ColumnID from crt, Value from crt)

Fig. 16

17/25

Algorithm: Updating a Row**Input(s):**

r = row number to update
row = set of (ColumnID, CellValue) pairs

Output(s):

none

Algorithm:

1. FOR EACH (col, val) in row set
 - 1.1 Find the appropriate Cell Repository Table (crt) for the column col
 - 1.2 Seek to row r in crt.
 - 1.3 UPDATE crt Value with val

Fig. 17

18/25

Numeric Cell Repository Table

Row#	Numeric ColumnID	Value
1	1	5123
1	2	18
1	3	20000
2	1	6471
2	2	26
2	3	42000
3	1	9880
3	2	28
3	3	41000
4	1	3362
4	2	22
4	3	28000

Updated Cell Value —

**Fig. 18**

19/25

Algorithm: Deleting a Row**Input(s):**

r = row number to delete

Output(s):

none

Algorithm:

1. FOR EACH Cell Repository Table (crt) in the Database containing row r :
 - 1.1 DELETE FROM crt WHERE Row# = r

Fig. 19

Numeric Cell Repository Table

Row#	Numeric ColumnID	Value
1	1	5123
1	2	18
1	3	20000
3	1	9880
3	2	28
3	3	39000
4	1	3362
4	2	22
4	3	28000

Numeric Column Definition Table

ColumnID	Column Name	Maximum
1	Employee#	9999
2	Age	150
3	Salary	100000

Fig. 20

String Cell Repository Table

Row#	String ColumnID	Value
1	1	Mark
1	2	Lismore
3	1	Dave
3	2	Sydney
4	1	Jane
4	2	Cairns

String Column Definition Table

ColumnID	Column Name	Length
1	Name	40
2	City	20

Fig. 20 (continued)

<p>Purpose of Query: Retrieve "Names" of Employees</p> <p>Query whereby information has been stored in a conventional manner (Figure @1)</p> <p>SELECT Name FROM EmployeeTable</p> <p>Equivalent query whereby information has been stored according to this invention (Figure @2)</p> <p>SELECT Value FROM StringCellRepositoryTable WHERE StringColumnID = 1</p>	<p>Purpose of Query: Calculate the average Age of Employees</p> <p>Query whereby information has been stored in a conventional manner (Figure @1)</p> <p>AVERAGE (SELECT Age FROM EmployeeTable)</p> <p>Equivalent query whereby information has been stored according to this invention (Figure @2)</p> <p>AVERAGE (SELECT Value FROM NumericCellRepositoryTable WHERE NumericColumnID = 2)</p>
<p>Purpose of Query: Retrieve "Names & Salaries" of Employees</p> <p>Query whereby information has been stored in a conventional manner (Figure @1)</p> <p>SELECT Name, Salary FROM EmployeeTable</p> <p>Equivalent query whereby information has been stored according to this invention (Figure @2)</p> <p>SELECT StringCellRepositoryTable.Value, NumericCellRepositoryTable.Value FROM StringCellRepositoryTable, NumericCellRepositoryTable WHERE StringColumnID = 1 AND NumericColumnID = 3</p>	<p>Purpose of Query: Count Employees from "Brisbane"</p> <p>Query whereby information has been stored in a conventional manner (Figure @1)</p> <p>COUNT (SELECT * FROM EmployeeTable WHERE City = "Brisbane")</p> <p>Equivalent query whereby information has been stored according to this invention (Figure @2)</p> <p>COUNT (SELECT * FROM StringCellRepositoryTable WHERE Value = "Brisbane")</p>

Fig. 21

Fig. 21 (continued)

22/25

Numeric Cell Repository Table

Row#	Numeric ColumnID	Value	Next Cell Value Location
1	1	5123	Numeric 1
1	2	18	Numeric 2
1	3	20000	String 1
2	1	6471	Numeric 1
2	2	26	Numeric 2
2	3	42000	String 1
3	1	9880	Numeric 1
3	2	28	Numeric 2
3	3	39000	String 1
4	1	3362	Numeric 1
4	2	22	Numeric 2
4	3	28000	String 1

String Cell Repository Table

Row#	String ColumnID	Value	Next Cell Value Location
1	1	Mark	String 2
1	2	Lismore	End
2	1	Mary	String 2
2	2	Brisbane	End
3	1	Dave	String 2
3	2	Sydney	End
4	1	Jane	String 2
4	2	Cairns	End

Fig. 22

23/25

Year Cell Repository Table (2-digit year values)

Row#	Year ColumnID	Value
1	1	93
2	1	61
3	1	65
4	1	94



*Year value cells for 2-digit year
representation regenerated to
store years as 4-digits*

Year Cell Repository Table (4-digit year values)

Row#	Year ColumnID	Value
1	1	1993
2	1	1961
3	1	1965
4	1	1994

Fig. 23

24/25

Date Cell Repository Table

Row#	Date ColumnID	Value
1	1	6/07/1997
1	2	6/09/2000
2	1	2/05/1994
2	2	1/03/1998
3	1	18/12/1997
3	2	18/3/2001
4	1	7/10/1996
4	2	1/01/1998

Date Column Definition Table

ColumnID	Column Name	Minimum
1	Appointed	1/01/1962
2	Next Holiday	1/01/1997

Fig. 24

25/25

Employee# Cell Repository Table

Row #	Employee#
1	5123
2	6471
3	9880
4	3362

Name Cell Repository Table

Row #	Name
1	Mark
2	Mary
3	Dave
4	Jane

Age Cell Repository Table

Row #	Age
1	18
2	26
3	28
4	22

City Cell Repository Table

Row #	City
1	Lismore
2	Brisbane
3	Sydney
4	Cairns

Salary Cell Repository Table

Row #	Salary
1	20000
2	42000
3	39000
4	28000

Fig. 25

INTERNATIONAL SEARCH REPORT

International Application No.
PCT/AU 98/00162

A. CLASSIFICATION OF SUBJECT MATTER

Int Cl⁶: G06F 017/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC: G06F 17/30, G06F 15/40

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US,A, 5423033 (YVEN) 6 June 1995 see whole document	
A	US,A, 5504890 (SANFORD) 2 April 1996 see whole document	
A	US,A, 5566330 (SHEFFIELD) 15 October 1996 see whole document	

☐ Further documents are listed in the
continuation of Box C

☐ See patent family annex

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier document but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search
12 May 1998

Date of mailing of the international search report
15 MAY 1998

Name and mailing address of the ISA/AU
AUSTRALIAN PATENT OFFICE
PO BOX 200
WODEN ACT 2606
AUSTRALIA
Facsimile No.: (02) 6285 3929

Authorized officer

S LEE

Telephone No.: (02) 6283 2205

PCT

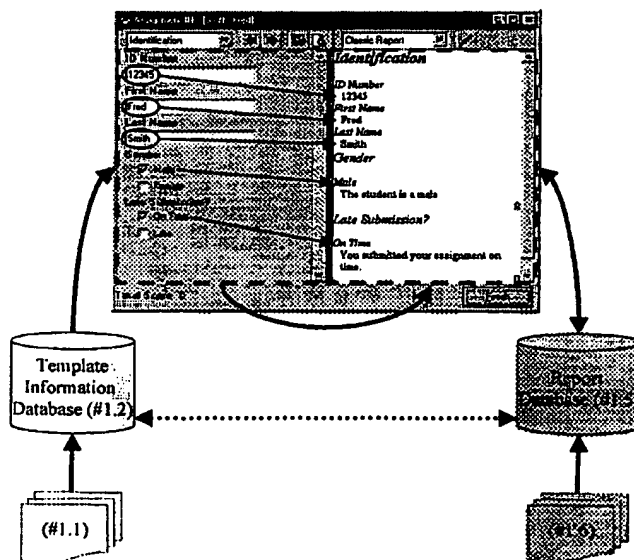
WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁶ : G06F 17/22		A1	(11) International Publication Number: WO 99/09492
			(43) International Publication Date: 25 February 1999 (25.02.99)
<p>(21) International Application Number: PCT/AU98/00649</p> <p>(22) International Filing Date: 14 August 1998 (14.08.98)</p> <p>(30) Priority Data: PO 8636 15 August 1997 (15.08.97) AU</p> <p>(71) Applicant (for all designated States except US): CLEVER-WORX INC. [US/AU]; Suite 168, 45 Cribb Street, Milton, QLD 4064 (AU).</p> <p>(72) Inventor; and</p> <p>(75) Inventor/Applicant (for US only): OLIVER, Brian, Keith [AU/AU]; 18/35 Troughton Road, Robertson, QLD 4109 (AU).</p> <p>(74) Agent: PIZZEYS PATENT & TRADE MARK ATTORNEYS; Level 6, 444 Queen Street, Brisbane, QLD 4000 (AU).</p>			<p>(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).</p> <p>Published With international search report.</p>

(54) Title: A COMPUTERISED METHOD OF COMPILING REPORTS



(57) Abstract

A method enabling the compilation of a report from stored template information, the method including: establishing a template database for storing user defined template information; providing presentation means for displaying stored template information whereby all or a portion of the template information may be selected for inclusion in a compiled report; establishing a report database to store reports generated by the assembly of all or portions of selected template information whereby each report may include a user defined set or sub-set of template information from the template database, and enabling the display of template information as well as one or more reports assembled from selected templates information.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

"A COMPUTERISED METHOD OF COMPILING REPORTS"

This invention relates to a computerised method of compiling a report from user defined template information.

This invention has particular application to such method whereby on-screen editing can occur during generation of the report created by the selection of stored template information. Preferably the editable report is assembled on-screen in one segment of a split window while the selected, editable and selectable template information for the report appears in another segment of a split window. Most preferably the report and the selected and selectable template information appear simultaneously at opposite sides of the screen.

BACKGROUND OF THE INVENTION

Much of what people and organisations do in the workplace is intimately tied up with the use, re-use and expansion of existing knowledge, expertise, facts, strategies and other information in reports, assessments and other documents. For example schoolteachers and university academics spend many hours marking and providing feedback on large numbers of student assignments. Doctors and other health professionals produce assessments or reports about patients that they deal with. Engineers, quality managers and auditors spend large amounts of time conducting inspections, audits or assessments and writing reports about each audit.

In each of these cases similar knowledge may be used to produce each report or assessment. For example, when marking student essays a teacher may want to provide feedback to students about the structure of their essays. One possible comment about the structure might be:

"Your essay does not include an adequate introduction.
Your introduction should at least summarise the main

arguments that you are going to use in the body of your essay."

Generally a teacher will hand write this comment onto the student's paper, or may type the feedback into a report. The teacher may write or type this comment many times. This is both time consuming and error prone. Furthermore effective expressions which may have been time consuming to compile are unlikely to be saved for future use.

In addition when apportioning marks it is often not possible to establish from the start a fair grading of marks. Often this can be not be done until all papers have been corrected. This may require a review of the papers which were first marked. Other grading tasks have similar difficulties.

This invention aims to alleviate at least one of the above disadvantages, and in one aspect, aims to provide means for making report generation tasks more efficient.

DESCRIPTION OF A COMPUTER

For the purposes of this application the term computer includes a machine of the general type as outlined in Fig. 6. Typically a computer optionally includes one or more selection devices being keyboard input or pointing devices, a plurality of display devices, mass storage means, printing means, network communication means, an input/output controller, main memory and a plurality of central processors and possibly associated cache memory, connected by a communication mechanism or bus of some description.

SUMMARY OF THE INVENTION

With the foregoing in view, this invention in one aspect resides broadly in a method enabling the compilation of an report from stored template information, the method including:

establishing a template database for storing user

defined template information;

providing presentation means for displaying stored template information whereby all or a portion of the template information may be selected for inclusion in a compiled report;

establishing a report database to store reports generated by the assembly of all or portions of selected template information whereby each report may include a user defined set or sub-set of template information from the template database, and

enabling the display of template information as well as one or more reports assembled from selected template information.

The template information may be pre-defined information but preferably the template information is user defined information and more preferably editable user defined information. It is also preferred that the display is enabled whereby simultaneous display of template information as well as one or more reports assembled from selected template information is provided and most preferably on a single screen and suitably having the template information displayed in one side of a split-window and the compiled report of the corresponding selected template information in the other side of a split-window. In addition it is preferred that the display of template information is searchable view of template information.

Preferably, the method includes:-

providing editing means for editing the template information and/or assembled reports;

establishing an association between the template and report databases whereby:

changes made to template information automatically update the corresponding unedited template information in assembled reports stored in the report database, and/or

changes made to report information in the report

database do not affect the corresponding template information in the template database. By this mechanism if a user selects a template information element for inclusion in a report, a corresponding element appears as a report information element in the report. Provided this corresponding report information remains unedited in the report, subsequent editing of the template information element will be reflected in the unedited corresponding report information element. Conversely subsequent editing of the report information element does not change the template information element.

A preferred method of associating template information contained within a report and stored in the report database, with the corresponding template information stored in the template database is to:-

- provide the template information as a structured set of information;

- uniquely identifying each element of the template information set stored in the template database, and

- storing only the corresponding template information identification in each compiled report stored in the report database. One preferred method of associating template and report information is to use conventional database "foreign" keys as outlined in Fig. 8. By using this method of association, changes made to the report database do not effect the template information, as only foreign key references are stored in the report database and not actual template information.

Furthermore any changes made to the template information may automatically be reflected in the report database as the report database foreign keys refer to the template information.

The method may also include:-

- establishing abbreviation means for abbreviating user defined template information, and

providing display customisation means whereby a user may choose to display either the template information or the abbreviated template information to minimise the amount of on-screen template information displayed. This enables more selectable template information elements from the template database to be simultaneously displayed for incorporation in full in the compiled report or reports.

In a preferred embodiment of the invention the method further includes:

providing associating means for associating user defined operational attributes with template information such that weightings may be attached as user-defined attributes to each template information, to allow the automatic generation of assessments score or grade or the like, along with the detailed report.

An efficient means for associating operational attributes to template information is to use conventional database technology to add an additional field or column of information to the template information database to store the appropriate operational attribute, as is the case with the weightings attribute in Fig 2, or by adding a BLOB (binary large object field) to store multiple operational attributes. By storing operational attributes with the template information in the template information database and using the above means of associating the template and report databases, changes made to an operational attribute, such as a weighting, will mean that all reports containing the weighted information will automatically be updated.

It is also preferred that the method includes constraining means for constraining the use of template information in a report based on previously selected information elements for the report. Preferably the template information to be constrained is user defined. By providing constraining means that respond to selection characteristics of the previously used template information in a report an

end-user will only be presented with and made available the relevant template information for their current report. Fig. 11 illustrates the result of constraining the available template information caused by the selection. This application of the constraints corresponds to the data in the "Constraint List" column of the Response Template Information Element Table of Fig. 2.

Constraints are beneficial, for example, when conducting surveys containing gender specific template information. Only the template information relevant to the gender selected will be displayed for inclusion in a compiled report.

It is further preferred that the above method includes a provision that allows an end-user to define and store the order in which template information will be presented for possible selection. This template information is illustrated by the "Scroll to ID" column in the Response Template Information Element table in Fig. 2.

By allowing an end-user to define "navigation information" and store it with the template information an end-user may simply indicate the next appropriate template information element displayed upon selection of some other previous template information. This allows for the "automatic scrolling" of template information to improve the efficiency of using the template information to compile editable reports. It also means an end-user does not have to use the standard scrolling mechanism to advance to the next selectable template information element.

It is further preferred that the above method of compiling reports also provides a means of exporting and communicating reports in user specified formats. For example, the template information could be a variety of media formats, including but not limited to text, print graphical, sound, animation or video. This may simply be achieved by sending the contents of the report split window or report database to one or more devices capable of producing the end-

user required format.

By utilising a method as defined above, a user may compile one or more reports each of which comprises a compilation of template information. Each compiled report may be displayed, played and/or edited as required. Any portion of template information in each of the reports may be changed automatically by making the desired change to the corresponding template information. An assessor using the above method may add, modify or remove particular template information in the template information database and have, if desired, all corresponding template information in previously compiled reports automatically updated without the need to individually select previously compiled reports to be updated. These additions, modifications or deletions to template information may take place before, during and after compilation of the reports.

Although the template and report databases may be realised as conventional databases or flat file structures or the like, this may not necessarily be the most efficient method of implementation. Preferably the template and/or report databases is realised as databases, hereinafter referred to ALPHA databases and being of the type described in our co-pending Australian patent application No. PCT\AU 98\00162.

This provides advantages over conventional databases structures in that user defined information can be more easily restructured and extended and in relation to flat file structures in that user defined information can be more easily and efficiently searched and managed, as required by this invention.

In another aspect, this invention resides broadly in a method enabling the compilation of a report from stored template information, the method including:

establishing a template database for storing user defined template information;

providing presentation means for displaying stored template information at one side of a split window whereby all or a portion of the template information may be selected for inclusion in a compiled report, and

providing managing means for simultaneously and selectively displaying all or portions of the stored template information at one side of a split window and the report or reports being assembled by selection of template information in the other side of said split window.

Preferably according to this further aspect the method includes:-

establishing a report database to store reports generated by the assembly of all or portions of selected template information whereby each report may include a user defined set or sub-set of template information from the template database;

providing editing means for editing the template information and/or assembled reports;

establishing an association between the template and report databases whereby:

changes made to template information automatically update the corresponding unedited template information in assembled reports stored in the report database, and/or

changes made to report information in the report database do not affect the corresponding template information in the template database.

The preferred forms of utilising the invention defined in pages 3 to 7 above are also preferred options in this further aspects of this invention defined below.

In yet another aspect this invention resides broadly in a method enabling the compilation of an editable report from stored editable template information, the method including:

establishing a template database for storing user defined template information;

providing selection means for displaying a searchable

view of stored template information whereby all or a portion of the selected template information may be selected for inclusion in a compiled report, and

establishing a report by the compilation of all or a portion of selected template information.

DESCRIPTION OF TYPICAL EMBODIMENT

In order that this invention may be readily understood and put into practical effect, reference will be made hereinafter to the accompanying drawings wherein:-

FIG. 1 is an overview of a typical embodiment of this invention;

FIG. 1A illustrates a further split screen illustrating the expanded display of a report from abbreviated template information;

FIG. 2 is an example of a template information database;

FIG. 3 is an illustration of a template information and compiled report split window;

FIG. 4 is an algorithm to display the template information database in a split window;

FIG. 5 illustrates a computer;

FIG. 6 specifies the characteristics of a computer;

FIG. 7 is an algorithm to display a compiled report constructed by selected template information;

FIG. 8 is an example of a report database;

FIG. 9 is an example template information "on selection" algorithm;

FIG. 10 is an example algorithm to "constrain" template information on selection;

FIG. 11 illustrates the result of constraining template information;

FIG. 12 is an example template information "on un-selection" algorithm;

FIG. 13 is an example algorithm to "de-constrain"

template information after un-selection;

FIG. 14 is an example algorithm to calculate a report score;

FIG. 15 illustrates an interface to modify template information properties (a);

FIG. 16 illustrates an interface to modify template information properties (b);

FIG. 17 illustrates an interface to modify template information properties (c), and

FIG. 18 illustrates an application user interface as an embodiment this invention.

According to the embodiment in Fig. 1 template information #1.1, Fig. 2 and Fig. 3 is stored in a template information database #1.2 and Fig. 2 and is retrieved and displayed according to the algorithm in Fig. 4 in a template window #1.3 and Fig. 3 for selection #3.1 by an end-user using a computer and selection device Figs. 5 and 6 to be included in a compiled report #3.2, Fig. 1A displayed according to the algorithm in Fig. 7 in a report window #1.4 and Fig. 3 and stored in a report database #1.5 and Fig. 8 as report records #1.6 and Figs. 8.

It should be noted that the database schema illustrated in Fig. 2 and Fig. 8 follow the structure of an ALPHA database as described in our co-pending Australian patent application No. PCT\AU 98\00162.

Preferably when an end-user selects template information #3.1 using a computer and selection device Figs. 5 and 6, the selection algorithm in Fig. 9 is performed along with the constraining algorithm in Fig. 10, Fig. 11 detailing the result of the constraint algorithm. The autoscroll algorithm is also performed when a user selects a template information element. This algorithm uses the value in the "Scroll to ID" column of the Response Template Information Element Table in Fig. 2 to determine the next template information element to display and then uses this new template information element

as the "first" value when applying the algorithm in Fig. 4.

Correspondingly, when an end-user un-selects template information #3.1 using a computer and selection device Figs. 5 and 6 the un-selection algorithm in Fig. 12 is performed along with the de-constraining algorithm in Fig. 13.

Furthermore whenever the compiled report is displayed through the use of the algorithm in Fig. 7, the report score should also be calculated, using the algorithm in Fig. 14, and displayed accordingly as in #3.3. A template information score #15.1, along with any other template information properties may be entered using the windows detailed in Figs. 15, 16 and 17 and stored as in Fig. 2. In particular, Abbreviated Template Information, Default Template Information Selection, Detailed Template Information and Template Information Constraints may be maintained by the end-user as in #15.2, #15.3, #16.1 and Fig. 17 respectively and stored in the template information database as in Fig 2. Fig. 17.1 details the currently constrained Template Information and Fig. 17.2 details the template information that may be constrained.

Fig. 18 outlines an application using a typical embodiment of this invention, detailing a structure of related template information elements #18.1, the tools to create new template information #18.2 and an assessment window #18.3, containing the selectable template information window and the compile report window.

The example below highlights the use of this invention for the purpose of academic assessment. Note however that the utility of this invention is not restricted to domain of academic assessment, rather this invention may be used in any field whereby information is gathered, collected, stored, analysed, reported or acted upon in some manner.

Example Scenario:

Assume an assessor is required to assess, score, grade and give feedback for a number of student assessments. During the assessment process, the assessor often recognises that many students make either similar mistakes or require similar feedback comments on their assignments. Rather than having to write the same or similar comments multiple times, the assessor may:

i). Define the common comments and feedback as template information Figs. 15, 16, 17 organised in a structure according to #18.1 in the template information split window #1.3, #3.1 and #18.3, and then

ii). during assessment simply select the required template information #3.1 in the template information split window #1.3 and #18.3 to have the system place the selected template information in the report split window #1.4, #3.2, and #18.3 where it may be edited and/or specialised #3.4 for the student as necessary.

By entering and selecting relevant template information in template information split window #3.1 for a student's assessment, a report is built up in the report split window 1.2 which may be later printed, faxed or emailed to the student as feedback on their assignment.

BENEFITS OF THIS INVENTION

This invention provides particular benefit in simplifying the process of creating reports where the content of the reports can be specified as template information as the content of the report is simply selected from template information, rather than being newly created for each report.

This invention makes template information visible and directly accessible to the user so that selection of the appropriate template information for each report is an efficient process.

This invention provides a mechanism for standardization

of the content of reports that are created by many different individuals. For example, where a large number of student assignments have to be marked, and these assignments are distributed among several teachers, this invention provides standardization of the criteria used to assess the students by making the same template information available to all assessors. This invention can be used to make specialist expert knowledge available to people who would not ordinarily possess this knowledge. For example, a professor may make detailed assessment criteria available as template information, and this template information may then be used by less knowledgeable staff in the assessment of student assignments.

A system implementing this invention allows the user of the system to see the report being constructed and to edit this report, as the user is working. In many other systems display and editing of compiled reports does not occur simultaneously with the construction of the report.

Because this invention allows additions, modifications and deletions to template information to be automatically applied to report information, this invention allows the template information to be improved over time without the need to regenerate existing reports. As a result this invention is particularly useful in industries where the content and structure of information to be collected, stored, compiled, processed and reported upon is not completely known or fixed when established.

For example, an industry that performs assessments, surveys, makes extensive use of notes, requires judgments to be made, or provides quotations, estimates and alike, could make use of systems implementing this invention. Similarly any field where the structure and content of information is volatile can also make extensive use of systems implementing this invention

It will of course be realised that the above has been

given only by way of illustrative example of the invention and that all such modifications and variations thereto as would be apparent to persons skilled in the art are deemed to fall within the broad scope and ambit of the invention as is defined in the appended claims.

THE CLAIMS DEFINING THIS INVENTION ARE AS FOLLOWS:-

1. A method enabling the compilation of an report from stored template information, the method including:

 establishing a template database for storing user defined template information;

 providing presentation means for displaying stored template information whereby all or a portion of the template information may be selected for inclusion in a compiled report;

 establishing a report database to store reports generated by the assembly of all or portions of selected template information whereby each report may include a user defined set or sub-set of template information from the template database, and

 enabling the display of template information as well as one or more reports assembled from selected template information.

2. A method as claimed in claim 1, including enabling simultaneous display of template information as well as one or more reports assembled from selected template information.

3. A method as claimed in claim 1 or claim 2, including providing editing means for editing the template information and/or assembled reports;

 establishing an association between the template and report databases whereby:

 changes made to template information automatically update the corresponding unedited template information in assembled reports stored in the report database, and/or

 changes made to report information in the report database do not affect the corresponding template information in the template database.

4. A method as claimed in any one of claims 1 to 3, including:

 providing the template information as a structured set of information;

 uniquely identifying each element of the template information set stored in the template database, and

 storing only the corresponding template information identification in each compiled report stored in the report database.

5. A method as claimed in any one of the preceding claims, including:

 establishing abbreviation means for abbreviating user defined template information, and

 providing display customisation means whereby a user may choose to display either the template information or the abbreviated template information to minimise the amount of on-screen template information displayed.

6. A method as claimed in any one of the preceding claims, including providing affiliation means for affiliating user defined operational attributes with template information.

7. A method as claimed in any one of the preceding claims, including:

 providing constraining means for constraining the use of template information to relevant template information based on previously selected template information for the report.

8. A method as claimed in any one of the preceding claims, wherein the presentation means provides a searchable view of stored template information and the report being assembled on one screen.

9. A method as claimed in any one of the preceding claims,

including:

providing editable navigation information as template information, and

providing navigation means utilising the provided navigation information to determine the next information element for display upon the selection of some other template information element by the user.

10. A method enabling the compilation of a report from stored template information, the method including:

establishing a template database for storing user defined template information;

providing presentation means for displaying stored template information at one side of a split window whereby all or a portion of the template information may be selected for inclusion in a compiled report, and

providing managing means for simultaneously and selectively displaying all or portions of the stored template information at one side of a split window and the report or reports being assembled by selection of template information in the other side of said split window.

11. A method as claimed in claim 10, including:

establishing a report database to store reports generated by the assembly of all or portions of selected template information whereby each report may include a user defined set or sub-set of template information from the template database;

providing editing means for editing the template information and/or assembled reports;

establishing an association between the template and report databases whereby:

changes made to template information automatically update the corresponding unedited template information in assembled reports stored in the report database, and/or

changes made to report information in the report database do not affect the corresponding template information in the template database.

12. A method as claimed in claim 10 or claim 11, including:

providing the template information as a structured set of information;

uniquely identifying each element of the template information set stored in the template database, and

storing only the corresponding template information identification in each compiled report stored in the report database.

13. A method as claimed in any one of claims 10 to 12, including:

establishing abbreviation means for abbreviating user defined template information, and

providing display customisation means whereby a user may choose to display either the template information or the abbreviated template information to minimise the amount of on-screen template information displayed.

14. A method as claimed in any one of claims 10 to 13, including providing affiliation means for affiliating user defined operational attributes with template information.

15. A method as claimed in any one of claims 10 to 14, including:

providing constraining means for constraining the use of template information to relevant template information based on previously selected template information for the report.

16. A method as claimed in any one of claims 10 to 15, wherein the presentation means provides a searchable view of

stored template information and the report being assembled on one screen.

17. A method as claimed in any one of claims 10 to 16, including:-

- providing editable navigation information as template information, and

- providing navigation means utilising the provided navigation information to determine the next information element for display upon the selection of some other template information element by the user.

18. A method enabling the compilation of an editable report from stored editable template information, the method including:

- establishing a template database for storing user defined template information;

- providing selection means for displaying a searchable view of stored template information whereby all or a portion of the selected template information may be selected for inclusion in a compiled report, and

- establishing a report by the compilation of all or a portion of selected template information.

19. A method as claimed in any one of claims 2, 10 or 18 including:

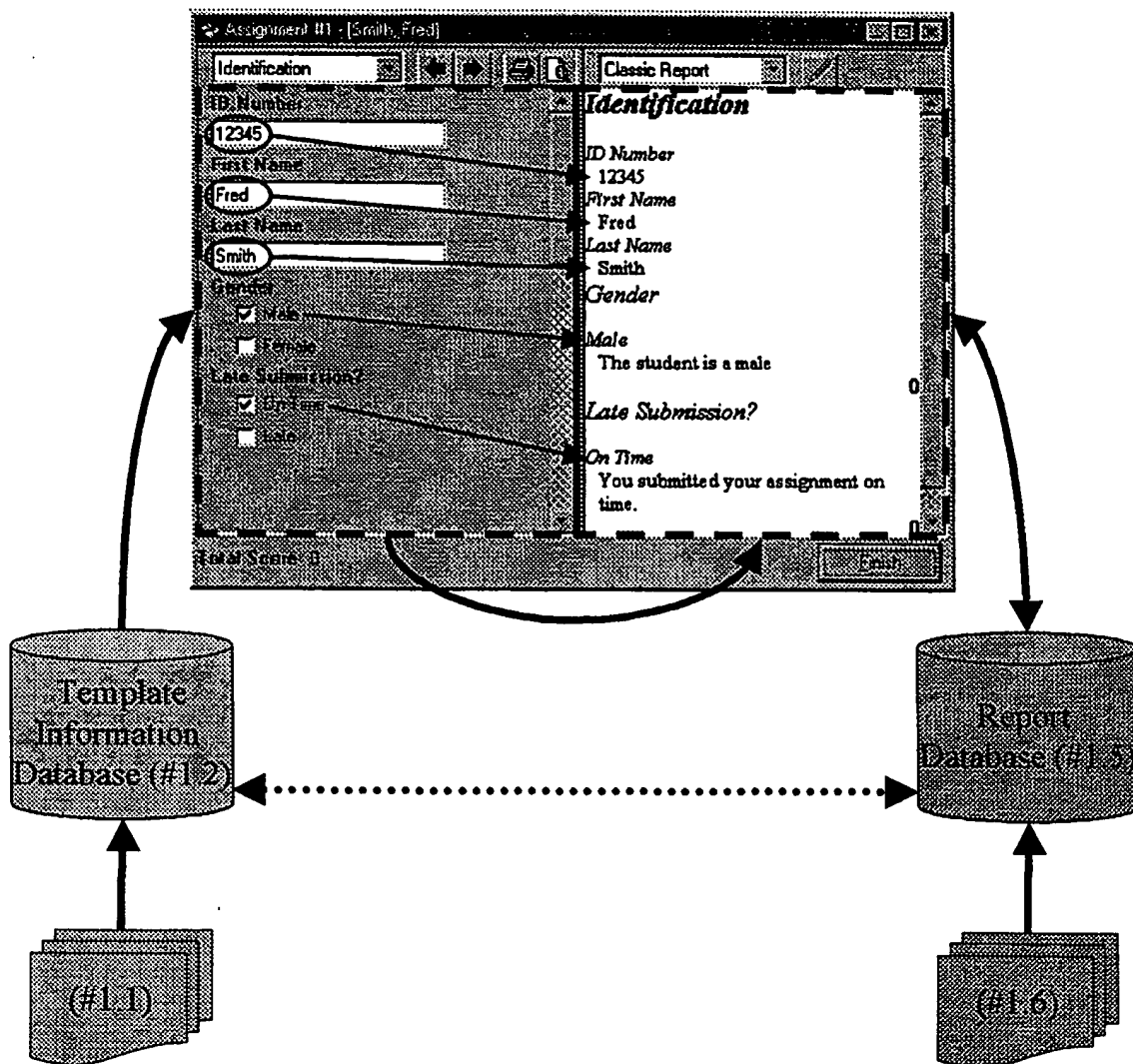
- providing editable weightings as template information, and

- providing arithmetic means utilising the provided weightings for calculating a weighted assessment score or grade or the like with the assembled report.

20. A method as claimed in any one of the proceeding claims, including:

- storing the template and/or report information in respective ALPHA databases.

1/19

Fig. 1

2/19

Fig. 1a

Presentation and Stru		Classic Report
Essay Structure <input checked="" type="checkbox"/> No Introduction <input type="checkbox"/> No Subheadings <input checked="" type="checkbox"/> No Bibliography or Acknowledgements...		Presentation and Structure Essay Structure <i>No Introduction</i> Your essay does not include an adequate introduction. Your introduction should at least summarise the main arguments that you are going to present in the body of your essay -5
Spelling <input type="checkbox"/> No spelling errors <input checked="" type="checkbox"/> 1 to 10 spelling errors <input type="checkbox"/> More than 10 spelling errors <small>Some spelling mistakes</small> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>		No Bibliography or Acknowledgements You have not included a Bibliography and you have not acknowledged your sources. It is very important that you do this so that the reader can tell which are your original ideas and which ideas came from someone else -5
		Spelling <i>1 to 10 spelling errors</i> Some spelling mistakes. I have not deducted any marks, but please take more care and use a dictionary to check your spelling.
		Finish

3/19

Fig. 2

Response Template Information Element Table

Template ID	Abbreviated Information	Detailed Information	Screen Position	Weighting	Select By Default	Constraint List	Scroll To ID	Other Attributes
5	Male	The student is male	1,5	1	False	-	7	-
6	Female	The student is female	1,6	2	True	-	7	-
8	On Time	Well Done...	1,8	2	True	-	10	-
9	Late	The assignment was..	1,9	-2	False	-	10	-
10	No Intro...	You did not provide an...	0,10	-2	False	11...15	-	-
12	Poor	A poor introduction	1,12	1	False	-	-	-
13	Average	An average introduction	1,13	2	False	-	-	-
14	Good	A good introduction	1,14	3	False	-	-	-
15	Excellent	An excellent introduction	1,15	4	False	-	-	-

Textbox Template Information Element Table

Template ID	Abbreviated Information	Detailed Information	Screen Position	Default Text	Other Attributes
1	Student #	Student Identification ...	0,0	#####	-
2	First Name	Student First Name...	0,1	Name	-
3	Last Name	Student Last Name	0,2	Name	-

Section Structure Template Information Element Table

Template ID	Abbreviated Information	Detailed Information	Screen Position	Child Template IDs	Other Attributes
4	Gender	Student Gender	0,4	5, 6	-
7	Late Sub..	Was submission late?	0,7	8,9	-
11	Introduction	Assignment Introduction	0,11	12..15	-

4/19

Fig. 3

Assignment #1 - [Smith, Fred]

Identification

ID Number
12345

First Name
Fred

Last Name
Smith

Gender
☒ Male
☐ Female

Late Submission?
☒ On Time
☐ Late

Classic Report

Identification

ID Number
12345

First Name
Fred

Last Name
Smith

Gender
Male
The student is a male

Late Submission?
On Time
You submitted your assignment on time.

Total Score 0

Finish

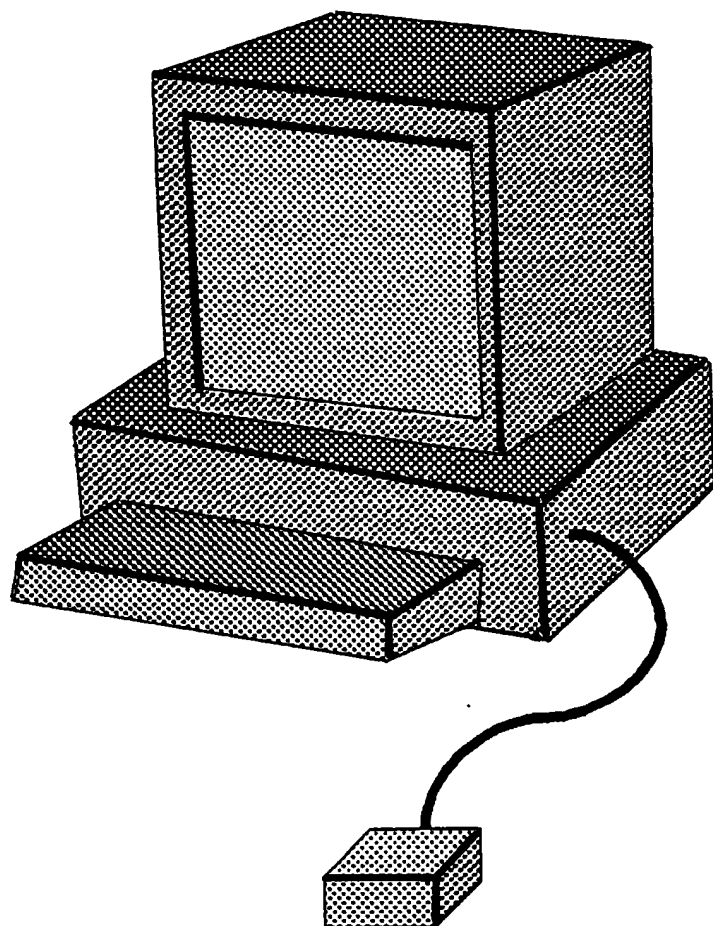
5/19

Fig. 4**Description:** Displays the template information in the Template Window**Input:** The first template information element (*first*) to display at the top of the Template Window**Returns:** Nothing*start drawing at the top of the template window**topPixel = 0**start with the specified first template information element**current = first**keep displaying while there are more template information elements that fit in the template window***while** topPixel < TemplateWindow.Height && current ≠ Nothing*should the current template information element be displayed?***if** current is constrained **then***are we showing constrained template information?***if** showing constrained template information **then***we are displaying constrained template information so display current as grey at topPixel**current.Display as Grey at topPixel**calculate the screen position of the next template information**topPixel = topPixel + current.Height + GAP***end if****else***the current template information is not constrained so display at topPixel**current.Display at topPixel**calculate the screen position of the next template information**topPixel = topPixel + current.Height + GAP***end if***step to the next template information after the current**current = current.Next***end while**

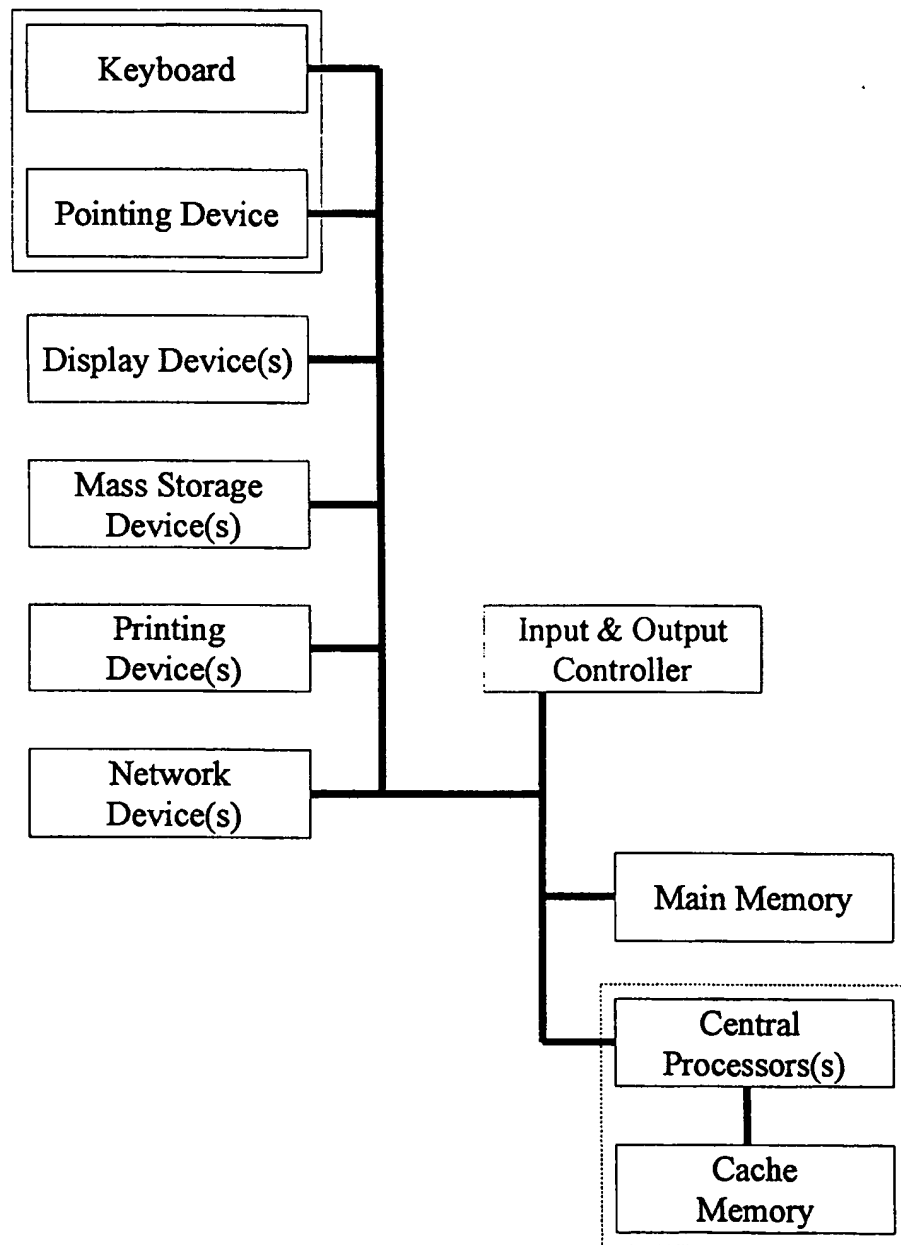
Calculated from the Screen Position in Template Information Database

6/19

Fig. 5



7/19

Fig. 6

8/19

Fig. 7**Description:** Displays the compiled report in the Report Window**Input:** The first selectable template information element (*first*) to display at the top of the Report Window**Returns:** Nothing*start drawing at the top of the report window*

topPixel = 0

*start with the specified first selectable template information element*current = *first**keep displaying while there are more selectable template information elements that fit in the report window***while** topPixel < ReportWindow.Height && current ≠ Nothing*is the current template information element selected?***if** current.Selected **then***is the currently selected template information element modified?***if** current.HasCustomInformation **then***display the modified template information element in the report window*

current.CustomInformation.Display at topPixel

else*display the standard template information in the report window*

current.Standard.Display at topPixel

end ifFrom the Report
TableFrom the
Template Table*calculate the screen position of the next selectable template information*

topPixel = topPixel + current.Height + GAP

end if*step to the next template information element after the current*

current = current.Next

end while*calculate and display report score***Display Report Score**

9/19

Fig. 8

Report Table for Response Element Information

Report ID	Template ID	Selected	Custom Information	Custom Weighting
1	5	True	-	-
1	6	False	-	-
1	8	True	Your assignment was early!	-
1	9	False	-	-
1	10	True	-	-
1	12	False	-	-
1	13	False	-	-
1	14	False	-	-
1	15	False	-	-

Report Table for Textbox Element Information

Report ID	Template ID	Text	Custom Information
1	1	12345	-
1	2	Fred	-
1	3	Smith	-

10/19

Fig. 9

Description: This algorithm is performed when template information is selected

Input: The template information element (*t*) selected

Returns: Nothing

constrain the template information as specified by the selected template information element

ConstrainTemplateInformation for *t*

does the selected template information element have an automatic scroll destination?

if *t* has an automatic scroll property **then**

set the first template information element in the template window to be the auto-scroll property

FirstTemplateInformation = *t*.AutoScrollTemplateInformation

end if

add the selected template information element to the report database

Add *t* to the report database

redraw the template information window (to refresh constraints applied and perform auto-scrolling)

Display Template Window

redraw the report window (to refresh the report)

Display Report Window

11/19

Fig. 10

Description: This algorithm is performed when template information element (*t*) is selected to constrain the remaining available template information

Input: The template information element (*t*) selected

Returns: Nothing

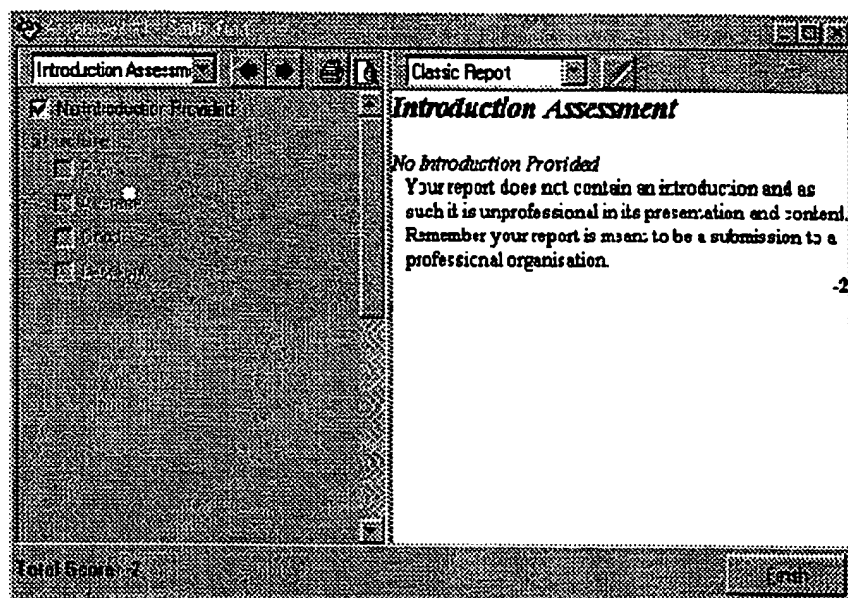
*constrain each of the template information elements specified by the
constrain list of template information element t.*

For Each Template Information Element (*j*) **in** *t*.ConstrainList **do**

constrain j due to the selection of t
j.constrainedBy t

End For Each

12/19

Fig. 11

13/19

Fig. 12

Description: This algorithm is performed when template information is unselected

Input: The template information element (t) unselected

Returns: Nothing

de-constrain the template information as specified by the unselected template information element. (Undo previous constraints due to the selection of t)

DeconstrainTemplateInformation for t

remove the unselected template information element from the report database

Remove t from the report database

redraw the template information window (to refresh constraints that no longer apply)

Display Template Window

redraw the report window (to refresh the report)

Display Report Window

14/19

Fig. 13

Description: This algorithm is performed when template information element (*t*) is unselected. Meaning previously constrained template information needs to be de-constrained.

Input: The template information element (*t*) unselected

Returns: Nothing

de-constrain each of the template information elements specified by the constrain list of template information element t.

For Each Template Information Element (*j*) in *t*.ConstrainList **do**

de-constrain j due to the unselection of t
j.deconstrainedBy t

End For Each

15/19

Fig. 14

Description: This algorithm is performed when a template information element is selected or unselected. It is used to return the score for a compiled report

Input: Nothing

Returns: Score for Report

Assume score is initially nothing
score = 0

For Each Selected Template Information Element (*t*) **in** Compiled Report **do**

add the score for selected template information element t which is the weighting of t multiplied by the weighting of t's parent template information
score = score + (*t*.Weighting * *t*.Parent.Weightig)

End For Each

return the calculated score
return score

16/19

Fig. 15

Response - Male

General | Help Definition | Referenced By | Comments | Hides | Reveal

Name:
Male

☐ Select by Default

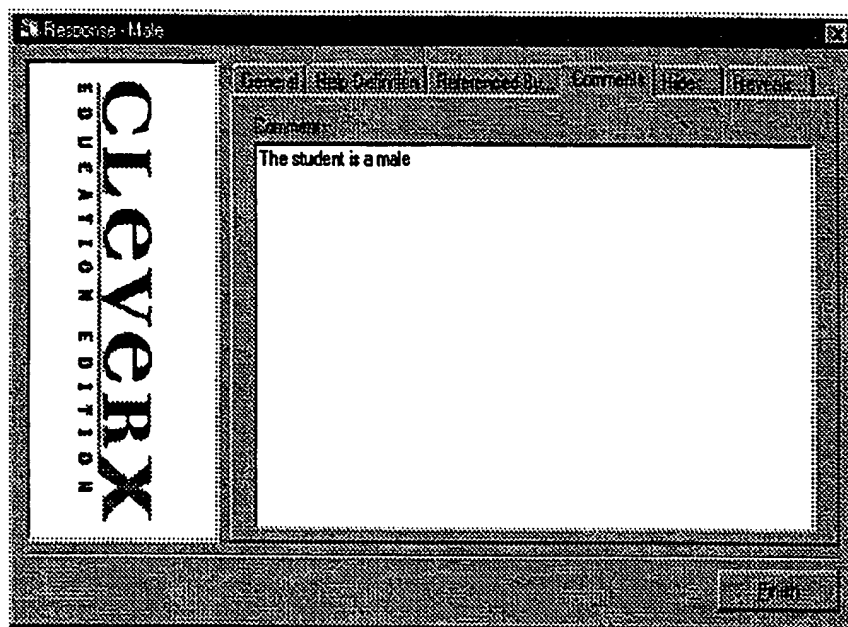
Score:
0

Exclusivity:
This is an exclusive response

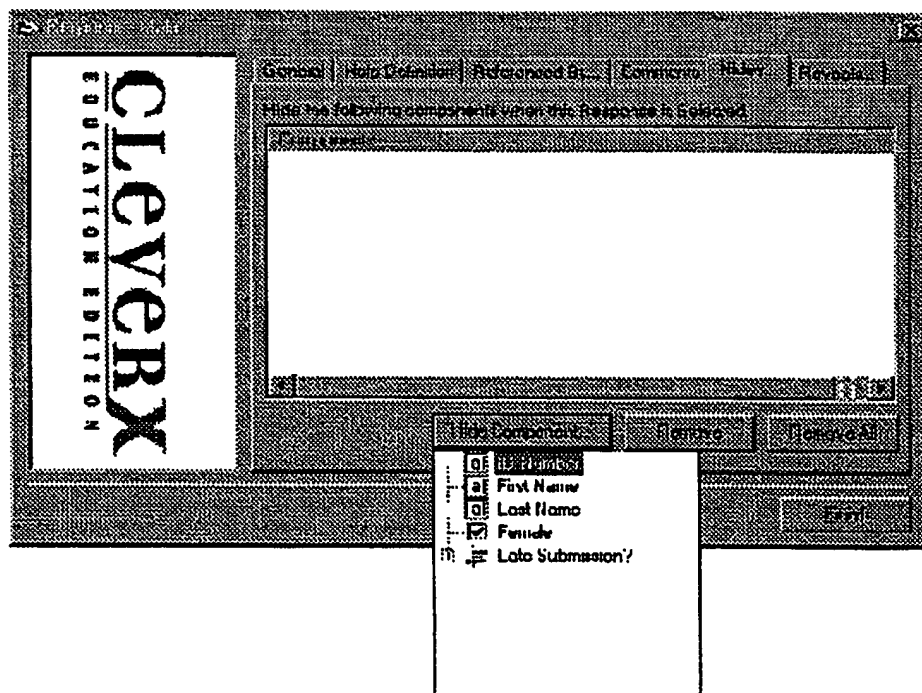
Statistics:
Times Selected: 1 (100.0%)
Times Unselected: 0 (0.0%)

Finish

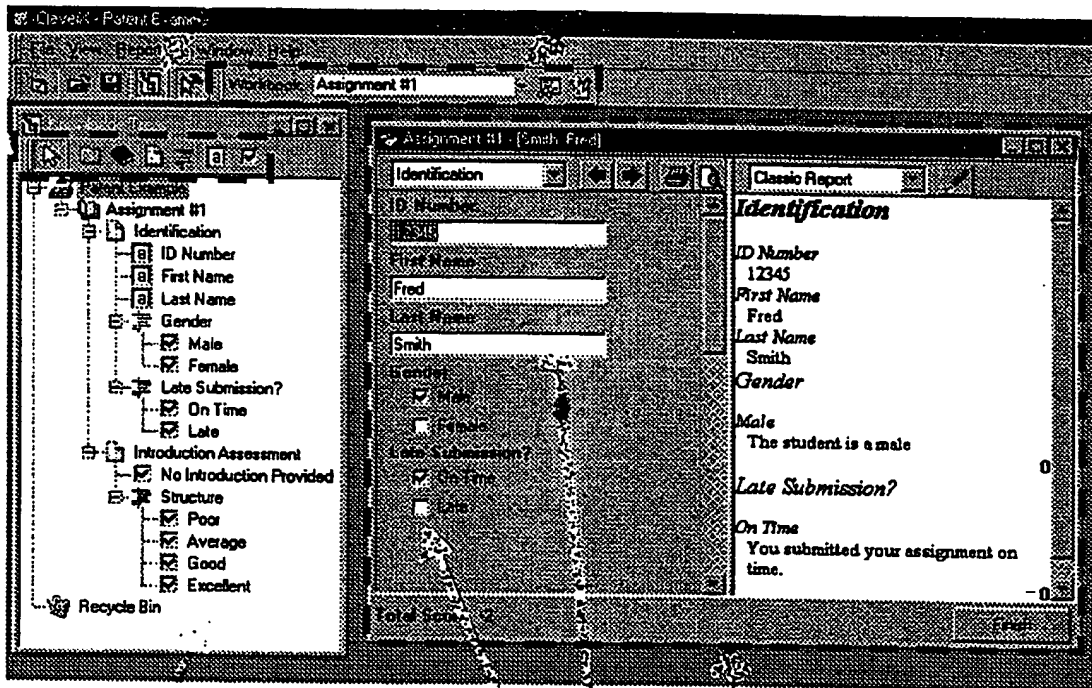
17/19

Fig. 16

18/19

Fig. 17

19/19

Fig. 18

INTERNATIONAL SEARCH REPORT

International Application No.
PCT/AU 98/00649

A. CLASSIFICATION OF SUBJECT MATTER																						
Int Cl ⁶ : G06F 17/22																						
According to International Patent Classification (IPC) or to both national classification and IPC																						
B. FIELDS SEARCHED																						
Minimum documentation searched (classification system followed by classification symbols) IPC G06F																						
Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched																						
Electronic data base consulted during the international search (name of data base and, where practicable, search terms used) WPAT G06F and ((Report: and template#) or (template# and database)), Window# and G06F and ((Report: and template#) or (template# and database) or (Report: and database))																						
C. DOCUMENTS CONSIDERED TO BE RELEVANT																						
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.																				
X	US, A, 5267155 (Buchanan et al.) 30 November 1993 see whole document	1-4,7-13,15-18																				
X	WO, A, 9634348 (Michael Umen & Company, Inc.) 31 October 1996 see whole document	1,10																				
<input type="checkbox"/> Further documents are listed in the continuation of Box C <input checked="" type="checkbox"/> See patent family annex																						
<p>* Special categories of cited documents:</p> <table border="0"> <tr> <td>"A"</td> <td>document defining the general state of the art which is not considered to be of particular relevance</td> <td>"T"</td> <td>later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention</td> </tr> <tr> <td>"E"</td> <td>earlier document but published on or after the international filing date</td> <td>"X"</td> <td>document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone</td> </tr> <tr> <td>"L"</td> <td>document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)</td> <td>"Y"</td> <td>document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art</td> </tr> <tr> <td>"O"</td> <td>document referring to an oral disclosure, use, exhibition or other means</td> <td>"&"</td> <td>document member of the same patent family</td> </tr> <tr> <td>"P"</td> <td>document published prior to the international filing date but later than the priority date claimed</td> <td></td> <td></td> </tr> </table>			"A"	document defining the general state of the art which is not considered to be of particular relevance	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention	"E"	earlier document but published on or after the international filing date	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone	"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art	"O"	document referring to an oral disclosure, use, exhibition or other means	"&"	document member of the same patent family	"P"	document published prior to the international filing date but later than the priority date claimed		
"A"	document defining the general state of the art which is not considered to be of particular relevance	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention																			
"E"	earlier document but published on or after the international filing date	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone																			
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art																			
"O"	document referring to an oral disclosure, use, exhibition or other means	"&"	document member of the same patent family																			
"P"	document published prior to the international filing date but later than the priority date claimed																					
Date of the actual completion of the international search 10 September 1998		Date of mailing of the international search report 16 SEP 1998																				
Name and mailing address of the ISA/AU AUSTRALIAN PATENT OFFICE PO BOX 200 WODEN ACT 2606 AUSTRALIA Facsimile No.: (02) 6285 3929		Authorized officer S. LEE Telephone No.: (02) 6283 2205																				

INTERNATIONAL SEARCH REPORT

International Application No.

PCT/AU 98/00649

This Annex lists the known "A" publication level patent family members relating to the patent documents cited in the above-mentioned international search report. The Australian Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

Patent Document Cited in Search Report				Patent Family Member			
US	5267155	AU	65096/90	CA	2067780	EP	496747
		US	5148366	WO	9106056		
WO	9634348	AU	55509/96	CA	2216822	EP	832462
		US	5734883				